공학박사 학위논문

하드웨어 기반의 고정확도 정수 단위 움직임 추정 및 병합 모드 추정

Hardware based High Accuracy Integer Motion Estimation and Merge Mode Estimation

2017 년 8 월

서울대학교 대학원

전기 컴퓨터 공학부

김 태 성

하드웨어 기반의 고정확도 정수 단위 움직임 추정 및 병합 모드 추정

Hardware based High Accuracy Integer Motion Estimation and Merge Mode Estimation

지도 교수 이 혁 재

이 논문을 공학박사 학위논문으로 제출함

2017 년 8 월

서울대학교 대학원

전기 컴퓨터 공학부

김 태 성

김태성의 공학박사 학위논문을 인준함

2017 년 8 월

위 원 장 채 수 익 (인)

부위원장 이 혁 재 (인)

위 원 최 기 영 (인)

위 원 조 남 익 (인)

위 원 이 채 은 (인)

초 록

HEVC는 H.264/AVC 대비 2배의 뛰어난 압축 효율을 가지지만, 많은 압축 기술이 사용됨으로써, 인코더 측의 계산 복잡도를 크게 증가시켰다. HEVC의 높은 계산 복잡도를 줄이기 위한 많은 연구들이 이루어졌지만, 대부분의 연구들은 H.264/AVC를 위한 계산 복잡도 감소 방법을 확장 적용하는 데에 그쳐, 만족스럽지 않은 계산 복잡도 감소 성능을 보이거나, 지나치게 큰 압축 효율 손실을 동반하여 HEVC의 최대 압축 성능을 끌어내지 못했다. 특히 앞서 연구된 하드웨어 기반의 인코더는 실시간 인코더의 실현이 우선되어 압축 효율의 희생이 매우 크다. 그러므로, 본 연구에서는 하드웨어 기반 Inter prediction의 고속화를 이룸과 동시에 HEVC가 가진 압축 성능의 손실을 최소화하고, 실시간 코딩이 가능한 하드웨어 구조를 제안하였다. 본 연구에서 제안한 bottom-up MV 예측 방법은 기존의 공간적, 시간적으로 인접한 PU로부터 MV를 예측하는 방법이 아닌, HEVC의 계층적으로 인접한 PU로부터 MV를 예측하는 방법을 제안하여 MV 예측의 정확도를 큰 폭으로 향상시켰다. 결과적으로 압축 효율의 변화 없이 IME의 계산 복잡도를 67% 감소시킬 수 있었다. 또한, 본 연구에서는 제안된 bottom-up IME 알고리즘을 적용하여 실시간 동작이 가능한 하드웨어 기반의 IME를 제안하였다. 기존의 하드웨어 기반 IME는 고속 IME 알고리즘이 갖는 단계별 의존성으로 인한 idle cycle의 발생과 참조 데이터 접근 문제로 인해, 고속 IME 알고리즘을 사용하지 않거나 또는 하드웨어에 맞게 고속 IME 알고리즘을 수정하였기 때문에 압축 효율의 저하가 수 퍼센트 이상으로 매우 컸다. 그러나 본 연구에서는 고속 IME 알고리즘인 TZS 알고리즘을 채택하여 TZS 알고리즘의 계산 복잡도 감소 성능을 훼손하지 않는 하드웨어 기반의 IME를 제안하였다. 고속 IME 알고리즘을 하드웨어에서 사용하기 위해서 다음 세 가지 사항을 제안하고 하드웨어에 적용하였다. 첫 째로, 고속 IME 알고리즘의 고질적 문제인 idle cycle 발생 문제를 서로 다른 참조 픽쳐와 서로 다른 depth에 대한 IME를 컨텍스트 스위칭을 통해 해결하였다. 둘 째로, 참조 데이터로의 빠르고 자유로운 접근을 위해 참조 데이터의 locality 이용한 multi bank SRAM 구조를 제안하였다. 셋 째로, 지나치게

자유로운 참조 데이터 접근이 발생시키는 대량의 스위칭 mux의 사용을 피하기 위해 탐색 중심을 기준으로 하는 제한된 자유도의 참조 데이터 접근을 제안하였다. 결과 제안된 IME 하드웨어는 HEVC의 모든 블록 크기를 지원하면서, 참조 픽처 4장를 사용하여, 4k UHD 영상을 60fps의 속도로 처리할 수 있으며 이 때 압축 효율의 손실은 0.11%로 거의 나타나지 않는다. 이 때 사용되는 하드웨어 리소스는 1.27M gates이다.

HEVC에 새로이 채택된 merge mode estimation은 압축 효율 개선 효과가 뛰어난 새로운 기술이지만, 매 PU 마다 계산 복잡도의 변동 폭이 커서 하드웨어로 구현되는 경우 하드웨어 리소스의 낭비가 많다. 그러므로 본 연구에서는 효율적인 하드웨어 기반 MME 방법과 하드웨어 구조를 함께 제안하였다. 기존 MME 방식은 이웃 PU에 의해 보간 필터 적용 여부가 결정되기 때문에, 보간 필터의 사용률은 50% 이하를 나타낸다. 그럼에도 불구하고 하드웨어는 보간 필터를 사용하는 경우에 맞추어 설계되어왔기 때문에 하드웨어 리소스의 사용 효율이 낮았다. 본 연구에서는 가장 하드웨어 리소스를 많이 사용하는 세로 방향 보간 필터를 절반 크기로 줄인 두 개의 데이터 패스를 갖는 MME 하드웨어 구조를 제안하였고, 높은 하드웨어 사용률을 유지하면서 압축 효율 손실을 최소화 하는 merge 후보 할당 알고리즘을 제안하였다. 결과, 기존 하드웨어 기반 MME 보다 24% 적은 하드웨어 리소스를 사용하면서도 7.4% 더 빠른 수행 시간을 갖는 새로운 하드웨어 기반의 MME를 달성하였다. 제안된 하드웨어 기반의 MME는 460.8K gates의 하드웨어 리소스를 사용하고 4k UHD 영상을 30 fps의 속도로 처리할 수 있다.

주요어 : High-efficiency video coding, Video compression, Integer motion estimation, merge mode estimation, hardware organization

학 번 : 2013-30230

목 차

표 목차

그림 목차

제 1 장 서 론

1.1 연구 배경

최근 스마트폰, 태블릿 PC 등이 널리 보급 되면서 동영상 컨텐츠의 제작 및 소비가 매우 활발하게 이루어 지고 있다. 동영상 컨텐츠의 제작은 인코딩, 소비는 디코딩 과정이 필수적으로 동반되는데 이러한 과정에는 반드시 표준화된 CODEC(enCOding/DECoding)이 필요하다. 과거 H.264/AVC(Advanced Video Coding)가 널리 이용되었으나, 디스플레이 장치의 급속한 발전에 따라, 고해상도 컨텐츠에 대한 수요가 증가하면서, 1920x1080 pixel의 Full high-definition (FullHD) 해상도를 갖는 동영상의 인코딩과 디코딩을 위해 설계된 H.264/AVC 표준으로 최근의 고해상도 영상의 수요에 효율적인 대응이 어렵게 되었다 [1]. 그러므로, 3840x2160 pixel의 해상도를 갖는 4k-UltraHD (UHD) 규격과 7680x4320 해상도를 갖는 8k-UHD 규격의 디스플레이 장치를 위한 새로운 동영상 압축 표준의 필요성이 대두되었고, 과거 H.264/AVC를 개발했던 Joint Video Team (JVT)의 Moving Picture Experts Group (MPEG)과 Video Coding Experts Group (VCEG)이 다시 새롭게 Joint Collaborative Team on Video Coding (JCT-VC)을 2010년에 결성하여 새로운 동영상 압축 표준을 개발 작업에 착수, 2013년도에 H.264/AVC의 압축 성능보다 우수한 새로운 동영상 압축 표준인 High-Efficiency Video Coding (HEVC)의 첫 draft를 발표하였다 [2].

High-Efficiency Video Coding (HEVC)는 4k- 및 8k-UHD 규격의 고해상도 영상의 코딩을 고려하여 설계되었고, H.264/AVC 대비 약 50%의 비트율로 동일한 화질의 영상을 제공할 수 있는 압축 성능을

가진다. HEVC의 이러한 압축 효율 능력의 향상은 H.264/AVC에서는 사용되지 않았던 새로운 압축 기술의 채택과, 기존 기술의 정교화에 따른 것으로 H.264/AVC 대비 인코더 측의 계산 복잡도가 큰 폭으로 증가하게 되었다. 주요 개선점은 다방면에 걸쳐 이루어졌다. 코딩 블록 단위가 매우 다양한 계층적 블록 구조를 가지며, intra prediction을 위해서는 35가지의 예측 모드를 사용하였다. Inter prediction을 위해서는 탭의 수가 증가한 보간 필터를 사용하였고, 기존의 Motion vector prediction 기반의 inter prediction에 더하여 merge mode가 새로이 추가되었다. Residual 코딩을 위한 transform에도 큰 변화가 이루어져, 기존 4x4와 8x8 블록 단위의 discrete cosine transform (DCT)에 더하여 16x16, 32x32 블록 단위의 DCT가 추가로 채택되었으며, DCT 뿐만 아니라 discrete sine transform (DST)도 사용되었다. 또한, 비트 스트림 으로부터 복원된 영상에 대해 적용되는 de-blocking 필터에 더하여, sample adaptive offset (SAO)를 추가로 적용하였다.

살펴본 바와 같이 HEVC는 H.264/AVC 대비 2배의 뛰어난 압축 효율을 가지지만, 많은 압축 기술이 사용됨으로써, 인코더 측의 계산 복잡도를 크게 증가시켰다. 과거 H.264/AVC가 처음 발표되었을 당시에도 높은 계산 복잡도 문제가 대두되어 계산 복잡도를 줄이기 위한 많은 연구들이 이루어졌고, HEVC 또한 마찬가지로, 발표 전후로 많은 연구들이 이루어졌다. 그러나, 대부분의 연구들은 H.264/AVC를 위한 계산 복잡도 감소 방법을 확장 적용하는 데에 그쳐, 만족스럽지 않은 계산 복잡도 감소 성능을 보이거나, 지나치게 큰 압축 효율 손실을 동반하여 HEVC의 최대 압축 성능을 끌어내지 못하고 있다. 이러한 문제점은 경우에 따라 과거 압축 기술인 H.264/AVC의 압축 성능 조차 달성하지 못하게 되어, 새로운 압축 표준 기술 사용의 의미를 퇴색시킬

수 있다. 특히 앞서 연구된 하드웨어 기반의 인코더는 실시간 인코더의 실현이 우선되어 압축 효율의 희생이 매우 크다. 그러므로, 본 논문에서는 하드웨어 기반 Inter prediction의 고속화를 이룸과 동시에 HEVC가 가진 압축 성능의 손실을 최소화하고, 실시간 코딩이 가능한 하드웨어 구조를 제안한다.

1.2 연구 내용

HEVC의 Inter prediction은 motion estimation (ME) 기반의 예측과 merge mode estimation (MME) 기반의 예측으로 이루어져 있다. ME는 과거 영상 압축 표준부터 널리 사용되어온 motion vector difference (MVD)와 motion vector predictor (MVP) 기반의 motion compensation (MC) 방법이다. 반면에 MME는 HEVC에 새로 채택된 방법으로, 오직 공간적, 시간적으로 인접한 PU의 움직임 정보만을 사용하여 예측과 MC를 수행하는 방법이다. 본 논문에서는 고해상도 영상에 대한 수요와 HEVC의 계층적 블록 구조로 인해 계산 복잡도가 크게 증가한, HEVC의 inter prediction을 위한 고속 알고리즘과 그 실시간 하드웨어 구현에 대해 다룬다.

첫째로, integer motion estimation (IME)을 위한 새로운 타입의 고속 알고리즘이 제안되었다. 제안된 고속 IME 알고리즘은 HEVC의 계층적 블록 구조에 착안하여, 계층간 MV의 유사성을 관찰하고 이를 기반으로 계층간의 MV 관계를 활용하여 IME의 계산 복잡도를 대폭 감소시킨다. 계층간 MV들은 그 유사성이 매우 높기 때문에, 이를 토대로 다른 계층의 MV를 IME를 수행하기 전에 예측할 수 있다. 예측된 MV는 예측하고자 하는 PU와 공간적으로 동일한 위치의, 그러나 계층이 다른

PU로부터 예측 되지 때문에, 그 정확도는 기존의 공간적, 시간적으로 인접한 PU로부터 MV를 예측하는 AMVP방식에 비하여 월등히 높다. 또한 HEVC의 계층적 블록 구조에 의해, 상위 계층으로부터 하위 계층을 예측하는 것 보다, 하위 계층으로부터 상위 계층을 예측하는 것이 더욱 정확하다. 이렇게 예측된 MV는 IME를 수행하고자 하는 PU의 IME의 초기 탐색 지점으로 사용되며, 높은 정확도를 활용하여 매우 좁은 영역에 대해서만 탐색이 이루어지더라도 정확한 MV를 찾아낼 수 있어서 뛰어난 계산 복잡도 감소 성능을 보인다.

둘째로, 제안된 고속 IME를 기반으로 한 하드웨어 기반 IME이 제안되었다. 비록 제안된 고속 IME의 계산 복잡도 감소 성능이 뛰어나지만, 실시간 IME를 위해서는 하드웨어를 통한 가속이 불가피하다. 또한 본 논문에 제안된 고속 IME를 포함한 많은 고속 IME 알고리즘들은 뛰어난 계산 복잡도 감소 성능을 가지지만, 그 알고리즘에 내재된 파이프라인과 병렬화를 방해하는 성질로 인해 하드웨어 기반의 IME에는 적합하지 않은 것으로 간주되어왔다. 이 때문에, 많은 하드웨어 기반의 IME는 파이프라인과 병렬화가 용이한 고속 IME 알고리즘을 사용하거나, 고성능의 고속 IME 알고리즘을 하드웨어에 맞도록 수정하여 사용해 왔지만, 이러한 고속 IME 알고리즘은 계산 복잡도 감소 성능이 본래의 성능보다 현저히 떨어진다. 그러나 본 논문에서는 고속 IME 알고리즘의 뛰어난 계산 복잡도 감소 성능을 유지하면서도 하드웨어 구현의 어려움을 극복한, 실시간 동작이 가능한 하드웨어 기반의 IME를 제안한다.

셋째로, HEVC의 새로운 inter prediction 방법인 MME를 위한 하드웨어 기반의 MME를 제안한다. MME는 IME에 비하여 그 계산 복잡도가 낮지만, 오히려 압축 효율에 미치는 영향이 크기 때문에 중요도가 높다.

MME는 계산 복잡도의 변동폭이 크고, 표준에 의해 그 동작이 명확하게 정해져 있기 때문에 효율적인 하드웨어 구현에 어려움을 가진다. 본 논문에서는 압축 효율의 손실을 최소화 하면서, 하드웨어의 사용률을 최대로 유지할 수 있는 하드웨어 기반의 MME를 제안한다.

넷째로, HEVC의 Inter Prediction 전체 관점에서 제안된 방법들을 적용한 CTU 단위의 3-stage 파이프라인 inter prediction 에 대해 다룬다. 또한 제안된 고속 IME 알고리즘이 기존의 고속 알고리즘과 가질 수 있는 conflict 에 대하여 논의하고, 제안된 고속 IME 알고리즘을 기존 알고리즘과 함께 사용할 수 있는 two-way 인코딩 방법을 제안한다.

마지막으로, 본 논문에서 제안된 방법들이 HEVC 이후의 새로운 영상 압축 표준에서 활용될 수 있는 방안에 대하여 모색하여 본다.

1.3 공통 실험 환경

본 논문에서 수행한 모든 실험들은 별도의 기재가 없는 경우 다음의 실험 환경을 따른다. 모든 실험은 HM 참조 소프트웨어를 사용하여 수행되었다. HM 참조 소프트웨어에 제공되는 Low delay P (LDP) 와 random access (RA)의 공통 실험 조건으로 실험이 수행되었다 [3]. 테스트 영상으로는 공통 실험 조건의 다섯 가지 클래스의 영상에 대해 실험이 이루어졌다. 이 영상들로는 클래스 A에서 2개, 클래스 B에서 다섯 개, 클래스 C에서 여섯 개, 클래스 D에서 3개, 클래스 E에서 세개의 영상이 있으며 해당 영상의 세부 내역은 다음과 같다. PeopleOnStreet, BasketballDrive, BQTerrace, Cactus, Kimono, ParkScene, Keiba, BQMall, BasketballDrill, Flowervase, PartyScene, RaceHorses, BasketballPass,

BlowingBubbles, RaceHorsesD, FourPeople, Johnny, KristenAndSara. 실험에 사용된 QP는 22, 27, 32, 37이다. 알고리즘의 적용으로 인한 압축 효율의 변화를 평가하기 위해서 Bjøntegaard delta bitrate (BD-BR)를 사용하였으며, 이 때 Y, U, V 컬러 성분에 각각 6:1:1의 비율의 가중치를 곱한 뒤 평균을 구하여 영상 전체의 BD-BR을 구하였다 [4].

1.4 논문 구성

본 논문의 나머지는 다음과 같이 구성된다. 2장에서는 HEVC의 inter prediction을 위한 고속화 알고리즘과 하드웨어 기반의 inter prediction에 대하여 다룬다. 3장에서는 IME의 계산 복잡도를 감소시키기 위한 bottom-up IME를 제안한다. 4장에서는 3장에서 제안한 bottom-up IME 알고리즘을 사용하여 하드웨어 기반의 실시간 IME를 구현할 때 발생하는 여러가지 문제점과 이를 해결하기 위한 방안을 함께 제시하여, 실시간 하드웨어 기반의 IME를 실현한다. 5장에서는 실시간 하드웨어 기반 MME의 구현을 위해 기존 MME의 하드웨어적인 비효율성에 대하여 논의하고, 그 비효율성을 개선한 하드웨어 기반의 MME를 제안한다. 6장에서는 앞서 다룬 하드웨어 기반의 IME와 MME를 사용하여 하드웨어 기반의 inter prediction을 구현 할 때에 고려되어야 하는 사항들에 대해서 논의하며, bottom-up IME이 가지는 top-down 기반의 고속 알고리즘들과의 conflict 문제에 대해 다루고 이를 해결하기 위한 two-way 인코딩을 제안한다. 7장에서는 HEVC를 위해 제안된 본 논문의 고속 알고리즘들을 차세대 영상 압축 표준에 적용할 수 있는지에 대한 가능성을 검토하여 본다. 마지막으로 8장에서 결론을 맺어 논문을 마친다.

제 2 장 관련 연구

2.1 HEVC 표준

2.1.1 쿼드-트리 기반의 계층적 블록 구조

HEVC는 압축 효율을 개선하기 위해 매우 유연한 계층적 블록 분할 구조를 채택하였다. 해당 블록 분할 구조는 coding tree unit (CTU), coding unit (CU), prediction unit (PU), transform unit (TU)로 분류된다. HEVC의 블록 분할 구조 예시를 그림 2.1에 나타내었다.

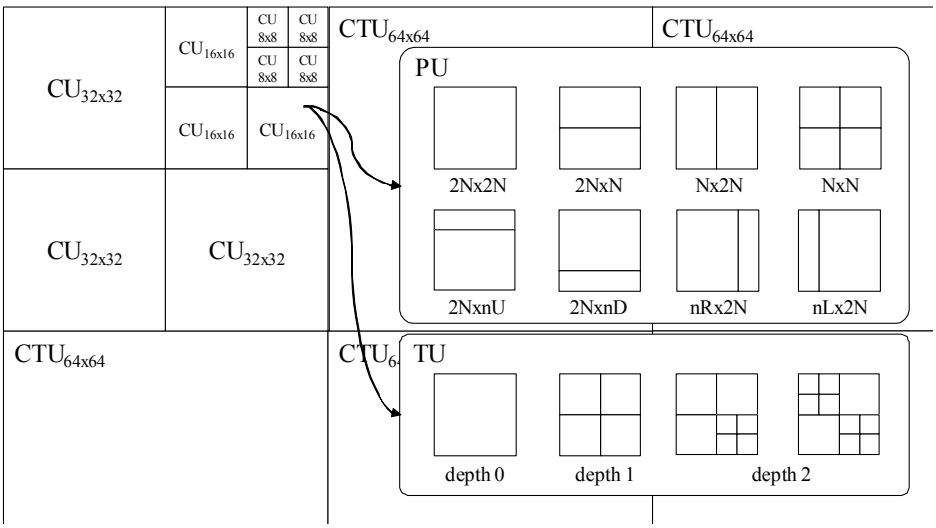


그림 2.1 HEVC의 계층적 블록 분할 구조

하나의 비디오 프레임은 여러 개의 슬라이스로 분할되며, 하나의 슬라이스는 다시 여러 개의 CTU로 분할된다. CTU는 H.264/AVC의 Macro block (MB)과 유사한 개념이지만, 그 크기가 16x16 pixel로 고정된 것이 아닌 16x16, 32x32, 64x64 pixel의 다양한 크기 중 하나가 선택 될 수 있다. CTU는 다시 쿼드-트리 구조로 분할될 수 있으며, 분할된 쿼드-트리의 가장 밑단을 CU라고 지칭한다. 분할 횟수에 따라서 CU의 크기가 결정되며, CTU는 최대 3회 까지 분할되어 64x64 CU, 32x32 CU, 16x16 CU, 8x8 CU의 다양한 CU 크기의 조합으로 CTU를 구성할 수 있다. 각 CU는 고유한 intra- 또는 inter- prediction의 예측 모드를 가질 수 있다. 또한 CU는 다시 그 내부에 Prediction unit (PU)와 Transform unit(TU) 단위의 분할이 가능하다. PU는 prediction을 수행하는 단위로 동일한 예측 파라미터를 사용하는 영역을 나타내고, TU는 prediction을 통해 생성된 residual을 코딩하는 단위로써, 동일한 TU에 속하는 영역에 대해서는 동일한 transform이 사용된다.

MB 단위로 intra-, inter prediction을 구분할 수 있었던 H.264/AVC와 다르게 HEVC의 블록 분할 구조는 CTU를 쿼드-트리 구조로 분할하여, 최대 64x64 pixel, 최소 8x8 pixel 크기의 CU로 블록을 구성할 수 있으므로, 넓은 영역에 대해서 예측 모드를 한번만 명시하여 비트를 절약하거나, 세밀한 영역 단위로 예측 모드를 결정하여, 보다 정확한 예측을 통해 residual 발생량을 줄일 수 있는 선택을 유연하게 결정할 수 있다.

PU와 TU 대해서 좀 더 자세히 살펴보면, PU는 CU의 크기를 2Nx2N으로 나타내었을 때, 총 8가지의 PU 분할이 가능하다. 2N×2N, 2N×N, N×2N, N×N, 2N×nU, 2N×nD, nR×2N, nL×2N의 총 8가지의 종류가 존재한다. 단, NxN PU는 1회 더 분할된 CU와 동일하므로, 더 이상

분할이 불가능한 최하위 계층의 CU에서만 사용된다. 또한 2N×nU, 2N×nD, nR×2N, nL×2N의 4 가지 PU 종류는 16x16 CU 이상 크기의 CU에서만 사용된다. 그러므로, 유연하게 분할된 CU 내부에서 다시 한번 PU로 분할될 수 있는 기회를 제공함으로써, 보다 세밀한 예측 파라미터 결정이 가능하다. 예측 파라미터로는 intra CU의 경우 예측 방향이 될 수 있으며, inter CU의 경우는 움직임 정보가 될 수 있다. TU는 CU 내부에서 CU와 유사하게 쿼드-트리 구조로 최대 2회까지 분할될 수 있으며 이 구조를 residual quad-tree (RQT)라고 부른다. TU에서 사용 가능한 transform의 크기는 32x32, 16x16, 8x8, 4x4 단위가 있으며, DCT가 사용되고, intra CU에 한해 DST가 사용 가능하다. PU와 TU는 서로 독립적인 분할 구조를 선택할 수 있으며, 단지 intra prediction이 사용된 CU에 한해 TU는 반드시 하나의 PU내에 속해야만 한다.

2.1.2 HEVC 의 Inter Prediction

Inter prediction에서는 참조 픽쳐리스트 내의 참조 픽쳐로부터 현재 예측 중인 PU와 가장 유사한 영역을 찾아, 해당 영역을 현재 PU를 위한 예측 픽셀로 활용한다. 해당 영역과 현재 PU 간의 공간적, 시간적 차이를 움직임 정보로써 비트스트림에 기록하여, 디코더에서 영상을 복원 할 때 이 움직임 정보를 사용하여 현재 PU를 위한 예측 픽셀의 위치를 특정하고, 예측 픽셀을 활용하여 복원을 수행할 수 있도록 한다. 그림 2.2에 inter prediction의 예를 나타내었다. 현재 픽쳐의 1, 2, 3 으로 표기된 사각형이 예측을 수행할 PU영역을 나타내며, 좌측에는 과거 영상인 참조 픽쳐 리스트 0, 우측에는 미래 영상인 참조 픽쳐 리스트 1을 나타낸다. HEVC에서 PU는 독립적인 예측 파라미터를 가질 수 있다.

그러므로, 매 PU는 서로 다른 움직임 정보를 가질 수 있다. 이 움직임 정보를 참조 픽쳐 리스트, 참조 픽쳐 인덱스, 움직임 벡터의 세가지로 이루어 진다. 하나의 PU는 한 개 또는 두개의 움직임 정보를 가질 수 있는데, 두개의 움직임 정보를 갖는 경우를 bi-prediction이라고 하며, 한 개의 움직임 정보를 갖는 경우를 uni-prediction이라고 한다. 그림 2.2의 1번 PU가 두 개의 움직임 정보를 갖는 bi-prediction으로 예측된 PU를 나타내며, 2, 3번 PU는 한 개의 움직임 정보를 갖는 uni-prediction으로 예측된 PU를 나타낸다.

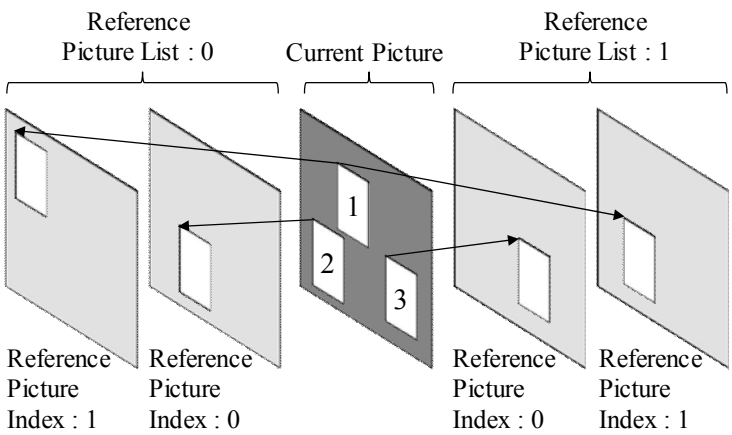


그림 2.2 HEVC의 inter prediction

매 PU마다 움직임 정보를 비트스트림에 기록해야 하기 때문에, HEVC에서는 움직임 벡터를 보다 적은 양의 비트로 기록 하기 위해서 두 가지의 기술을 채택하였다. 한 가지는 adaptive motion vector prediction

(AMVP)이며, 다른 한 가지는 merge mode prediction이다. AMVP는 현재 예측 중인 PU의 공간적으로 또는 시간적으로 인접한 PU들로부터 현재 PU의 MV를 예측하여, 단지 예측 오차 만을 비트스트림에 기록하도록 함으로써, MV를 그대로 비트스트림에 기록하는 것 보다 적은 비트로 움직임 정보를 디코더로 전송할 수 있도록 하는 기술이다. 반면에, HEVC에 처음으로 채택된 merge mode prediction은 AMVP와 유사하지만, MV 뿐만 아니라, 참조 픽쳐 리스트, 참조 픽쳐 인덱스도 이웃 PU로부터 예측하며, MV 예측 오차의 보정을 허용하지 않음으로써, 움직임 정보를 위해 사용되는 비트를 더욱 감소시킨 기술이다. 즉, merge mode로 예측된 PU는 merge mode 사용 여부를 나타내는 merge flag와, 이웃 PU중 어떤 PU와 병합되었는지 나타내는 merge index의 단순한 정보만으로, 현재 PU의 움직임 정보를 나타내게 된다

.

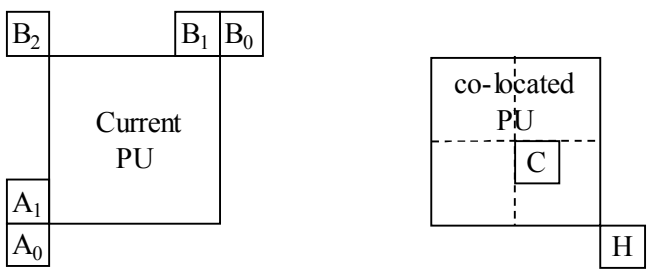


그림 2.3 AMVP와 Merge mode가 사용하는 공간적, 시간적으로 인접한 PU의 위치

AMVP에 의해 MV를 예측 할 때는 공간적 및 시간적으로 인접한 PU로부터 예측에 필요한 정보를 얻는다. AMVP는 PU가 사용할 수 있는

매 참조 픽쳐에 대해서 매번 수행된다. 그림 2.3에 AMVP와 merge mode가 움직임 정보를 예측할 때 사용하는 공간적, 시간적으로 인접한 PU의 위치를 나타내었다. 그림 2.3 (a)는 다섯 개의 공간적으로 인접한 PU의 위치를 나타내며, 그림 2.3 (b)는 두 개의 시간적으로 인접한 PU의 위치를 나타낸다. A0, A1, B0, B1, B2의 로 표시된 곳에 위치한 현재 픽쳐의 PU들이 공간적으로 인접한 PU이다. 또한 슬라이스 헤더에 명시된 co-located 참조 픽쳐 내에서 C와 H로 표시된 곳에 위치한 PU가 시간적으로 인접한 PU 이며, 이 PU를 co-located PU로 부른다. 먼저, 좌측에 위치한 A0와 A1의 위치에 대해서 사용 가능한 MV가 있는지 순차적으로 확인하여 공간적으로 좌측에 인접한 PU로부터 하나의 사용 가능한 MV를 MVP로 사용한다. 다음으로 상단에 위치한 B0, B1, B2에 위치한 PU들을 순차적으로 사용 가능한 MV가 있는지 확인한다. 이 경우에도 마찬가지로 사용 가능한 MV가 있는 경우 가장 먼저 확인한 MV를 MVP로 사용한다. 앞선 과정에서 2개의 MVP가 유도된 경우, 2개의 MVP가 동일한 것이라면 나중에 유도된 것을 삭제한다. 만약 공간적으로 인접한 PU로부터 2개의 MVP를 유도하지 못했다면, 시간적으로 인접한 PU로부터 MVP를 유도하게 된다. 시간적으로 인접한 PU는 H와 C의 순서대로 확인하게 되며, H 위치의 PU의 MV가 사용가능하지 않다면 C의 위치를 확인하며, 두 위치의 MV를 모두 MVP로 사용하지는 않는다. 공간적, 시간적으로 인접한 PU의 움직임 정보의 참조 픽쳐 리스트와 참조 픽쳐 인덱스가 현재 PU가 사용하는 참조 픽쳐 리스트와 참조 픽쳐 인덱스와 다른 경우, 참조 픽쳐의 시간적 위치관계에 따라 MV를 스케일하여 사용하게 된다. 만약 공간적, 시간적으로 인접한 PU로부터 2개의 MVP를 유도하지 못한 경우, 움직임이 없는 zero MV로 결손 된 MVP를 대체하게 된다. 이렇게 구성된

AMVP 리스트에는 언제나 2개의 MVP가 존재하며, ME를 통해 찾아진 MV와의 오차가 작은 MVP를 최종 MVP로 선택하여 사용하게 된다.

Merge mode 또한 그림 2.3에 나타낸 공간적, 시간적으로 인접한 PU로부터 움직임 정보를 예측한다. 단지, 현재 PU가 사용하는 참조 픽쳐마다 독립적으로 MVP를 유도하는 AMVP와 다르게, 참조 픽쳐 리스트와 참조 픽쳐 인덱스를 포함한 모든 움직임 정보를 이웃 PU로부터 예측하기 때문에 각 PU 별로 한 번씩 수행된다. 또한 공간적으로 인접한 다섯 개의 PU 들로부터 최대 4개의 merge mode 후보를 유도할 수 있으며, 시간적으로 인접한 PU들로부터는 한 개의 후보를 유도할 수 있다. Merge mode는 최대 다섯 개의 후보를 가질 수 있기 때문에, 공간적 시간적으로 인접한 PU들로부터 다섯 개의 후보가 유도되지 못한 경우, 앞서 유도된 후보를 조합하는 bi-predictive 후보를 유도하며, 그럼에도 불구하고 다섯 개의 후보를 모두 만들어내지 못하 경우에는 움직임이 없는 zero MV로 결손된 후보를 대체하여 merge mode 후보 리스트를 생성한다.

Motion compensation (MC)을 위해서 HEVC는 half-, quarter- pixel을 생성하기위해 luma 성분을 위해 8-tap, chroma 성분을 위해 4-tap 의 보간 필터를 사용하였다. 이는 6-tap 및 bi-linear filter를 사용한 H.264/AVC의 보간 필터보다 복잡도가 크게 증가한 것이다. 그림 2.4에 HEVC의 Luma 성분을 위한 보간 필터를 나타내었다. 그림 2.4에서 대문자 A로 표시된 부분이 정수 단위 픽셀을 나타내며, 소문자로 표시된 부분이 소수 단위 픽셀을 나타낸다. Luma 성분을 위한 보간 필터는 크게 세 가지가 있으며 half 픽셀 생성을 위한 한 가지의 보간 필터와 quarter 픽셀 생성을 위한 두 가지의 보간 필터가 존재한다. 그림의 좌측에 세 가지의 필터가 적용될 픽셀의 위치와 함께, 해당 보간 필터의 tap의 계수 및 필터가

적용 될 인근 픽셀의 인덱스를 함께 나타내었다. Half 픽셀을 위해서 8-tap의 보간 필터가 사용되며, quarter 픽셀을 위해서 7-tap의 보간 필터가 사용된다.

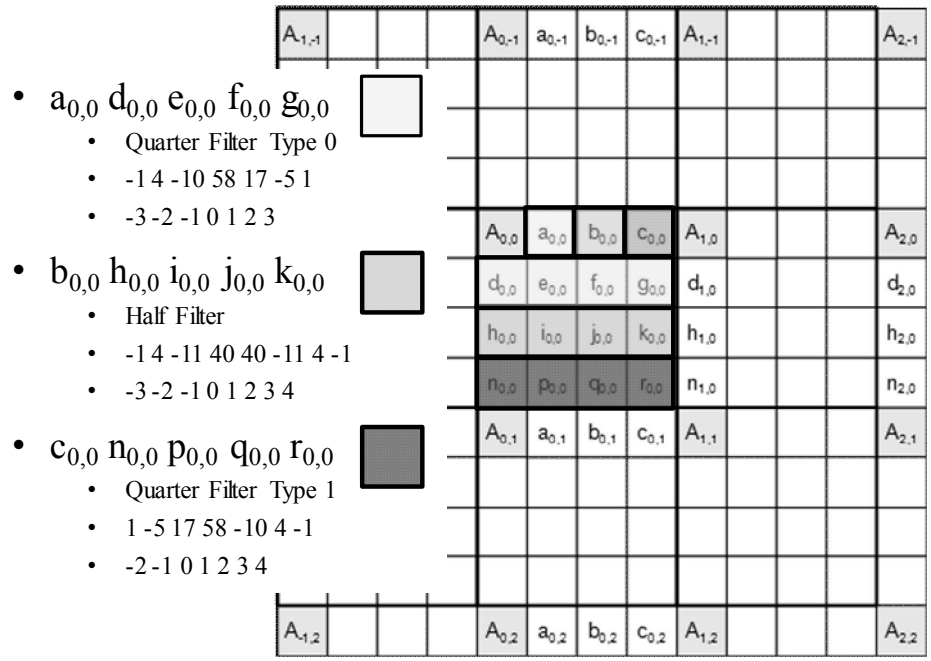


그림 2.4 HEVC의 Luma 성분을 위한 보간 필터

표 2.1에 그림 2.4에 나타낸 HEVC의 Luma성분을 위한 보간 필터의 연산을 모든 소수 단위 픽셀 위치에 대하여 상세히 나타내었다. 그림 2.4의 a, b, c, d, h, n 위치의 소수 단위 픽셀에 대해서는 정수 단위 픽셀을 사용하여 보간 필터가 이루어지며, 나머지 소수 단위 픽셀 위치에 대해서는 a, b, c 위치의 소수 단위 픽셀을 사용하여 보간 필터가

이루어진다. Clip 연산은 보간 필터 적용 결과를 반올림 처리하여 시프트 연산을 수행한 다음, 8 bit의 픽셀 값의 범위를 벗어난 경우 최소 또는 최대 값으로 clip 하는 연산이다. 주의해야 할 점은 보간 필터를 가로 및 세로 각 방향에 대해 모두 적용해야 하는 경우, clip 연산을 수행하지 않은 1차 보간 필터 결과를 그대로 2차 보간 필터의 입력으로 사용해야 한다는 것이다.

표 2.1 HEVC의 Luma 성분을 위한 보간 필터 연산

HEVC interpolation filter operation for luma component
$a'_{0,0} = -A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 58*A_{0,0} + 17*A_{1,0} - 5*A_{2,0} + A_{3,0}$
$b'_{0,0} = -A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0}$
$c'_{0,0} = A_{-2,0} - 5*A_{-1,0} + 17*A_{0,0} + 58*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0}$
$d'_{0,0} = -A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 58*A_{0,0} + 17*A_{0,1} - 5*A_{0,2} + A_{0,3}$
$h'_{0,0} = -A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4}$
$n'_{0,0} = A_{0,-2} - 5*A_{0,-1} + 17*A_{0,0} + 58*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4}$
$e'_{0,0} = -a'_{0,-3} + 4 * a'_{0,-2} - 10 * a'_{0,-1} + 58 * a'_{0,0} + 17 * a'_{0,1} - 5 * a'_{0,2} + a'_{0,3}$
$i'_{0,0} = -a'_{0,-3} + 4 * a'_{0,-2} - 11 * a'_{0,-1} + 40 * a'_{0,0} + 40 * a'_{0,1} - 11 * a'_{0,2} + 4 * a'_{0,3} - a'_{0,4}$
$p'_{0,0} = a'_{0,-2} - 5 * a'_{0,-1} + 17 * a'_{0,0} + 58 * a'_{0,1} - 10 * a'_{0,2} + 4 * a'_{0,3} - a'_{0,4}$
$f'_{0,0} = -b'_{0,-3} + 4 * b'_{0,-2} - 10 * b'_{0,-1} + 58 * b'_{0,0} + 17 * b'_{0,1} - 5 * b'_{0,2} + b'_{0,3}$
$j'_{0,0} = -b'_{0,-3} + 4 * b'_{0,-2} - 11 * b'_{0,-1} + 40 * b'_{0,0} + 40 * b'_{0,1} - 11 * b'_{0,2} + 4 * b'_{0,3} - b'_{0,4}$
$q'_{0,0} = b'_{0,-2} - 5 * b'_{0,-1} + 17 * b'_{0,0} + 58 * b'_{0,1} - 10 * b'_{0,2} + 4 * b'_{0,3} - b'_{0,4}$
$g'_{0,0} = -c'_{0,-3} + 4 * c'_{0,-2} - 10 * c'_{0,-1} + 58 * c'_{0,0} + 17 * c'_{0,1} - 5 * c'_{0,2} + c'_{0,3}$
$k'_{0,0} = -c'_{0,-3} + 4 * c'_{0,-2} - 11 * c'_{0,-1} + 40 * c'_{0,0} + 40 * c'_{0,1} - 11 * c'_{0,2} + 4 * c'_{0,3} - c'_{0,4}$
$r'_{0,0} = c'_{0,-2} - 5 * c'_{0,-1} + 17 * c'0,0 + 58 * c'_{0,1} - 10 * c'_{0,2} + 4 * c'_{0,3} - c'_{0,4}$
Clipping operation
$a_{0,0} = \text{Clip}(0,255,(a'_{0,0} + 2^5) >> 6)$
$b_{0,0} = \text{Clip}(0,255, (b'_{0,0} + 2^5) >> 6)$
$c_{0,0} = \text{Clip}(0,255, (c'_{0,0} + 2^5) >> 6)$
$d_{0,0} = \text{Clip}(0,255, (d'_{0,0} + 2^5) >> 6)$
$h_{0,0} = \text{Clip}(0,255, (h'_{0,0} + 2^5) >> 6)$
$n_{0,0} = \text{Clip}(0,255, (n'_{0,0} + 2^5) >> 6)$
$e_{0,0} = \text{Clip}(0,255, (e'_{0,0} + 2^{11}) >> 12)$
$i_{0,0} = \text{Clip}(0,255, (i'_{0,0} + 2^{11}) >> 12)$
$p_{0,0} = \text{Clip}(0,255, (p'_{0,0} + 2^{11}) >> 12)$
$f_{0,0} = \text{Clip}(0,255, (f'_{0,0} + 2^{11}) >> 12)$
$j_{0,0} = \text{Clip}(0,255, (j'_{0,0} + 2^{11}) >> 12)$
$q_{0,0} = \text{Clip}(0,255, (q'_{0,0} + 2^{11}) >> 12)$
$g_{0,0} = \text{Clip}(0,255, (g'_{0,0} + 2^{11}) >> 12)$
$k_{0,0} = \text{Clip}(0,255, (k'_{0,0} + 2^{11}) >> 12)$
$r_{0,0} = \text{Clip}(0,255, (r'_{0,0} + 2^{11}) >> 12)$

2.2 화면 간 예측의 속도 향상을 위한 이전 연구

2.2.1 고속 Integer Motion Estimation 알고리즘

Integer motion estimation (IME)는 HEVC의 inter prediction을 위해서 현재 PU와 가장 유사한 정수 단위 픽셀 영역을 참조 픽쳐 내에서 탐색하는 과정이다. IME의 계산 복잡도는 영상의 해상도에 따라서 증가하게 된다. 이는 영상의 크기가 커지면 코딩해야하는 픽셀의 수도 증가하여 IME의 연산량 또한 자연스럽게 증가하게 되기 때문이다. 또한, 동일한 내용의 영상이라고 하더라도, 저해상도 영상보다 고해상도 영상에서는 더욱 큰 픽셀 단위의 움직임을 가지므로 고해상도의 영상을 코딩할수록 더욱 넓은 범위에 대해 IME를 수행해야만 정확한 MV를 찾아낼 수 있다. 때문에 계산 복잡도를 줄이기 위한 고속 IME 알고리즘에 대한 연구는 오늘날에도 이루어지고 있다. 고속 IME 알고리즘에 대한 연구를 크게 두 가지로 분류하면, IME를 시작하기 이전에 MV를 예측하여 IME를 수행할 기준점을 정하는 연구와, 주어진 IME의 시작 지점으로부터 탐색을 수행하여 보다 적은 계산 복잡도로 실제 MV를 빠르게 찾아내기 위한 연구가 있다.

첫번째 분야는, IME를 시작하기 전에 최적의 MV를 예측하는 연구들이며 decoder-side motion vector derivation (DMVD)에 관한 연구와 그 맥을 같이 하고 있다. DMVD에 관한 연구들은 적은 양의 비트로 MV를 전송하기 위해, MVP와 MV간의 오차인 motion vector difference (MVD)를 최소화 하기 위해 노력해왔다 [5-8], [57]. 그런데 IME는 MVP 위치로부터 탐색을 시작하므로, 결과적으로, 정확한 MVP를 사용하여 수행된 IME는 적은 계산 복잡도만으로도 실제 MV에 빠르게 수렴할 수 있다. 그러므로

정확한 MVP는 MVD의 전송에 사용되는 비트를 줄여 보다 높은 코딩 효율을 달성하기 위함 뿐만 아니라, IME에 있어서 계산 복잡도를 감소시키는 데에 중요한 역할을 한다. HEVC에 채택된 AMVP는 DMVD의 대표적인 연구 중 하나로 HM 참조 소프트웨어의 IME의 시작 지점으로도 활용되고있다 [3],[5].

두번째 분야는, block matching algorithm (BMA)를 기반으로 가장 활발하게 연구가 이루어진 분야로써 보다 적은 계산 복잡도로 실제 MV를 찾아내기 위한 노력이 이루어졌다. BMA는 현재 블록과 가장 유사한 블록을 찾을 때 현재 블록과 후보가 되는 다수의 블록들을 비교하여 가장 유사한 블록을 찾아내는 방법이다. IME에서는 여러 장의 참조 픽쳐 내에서 BMA를 수행하여 현재 PU를 위한 MV를 찾는다. 그러므로 보다 적은 수의 위치에 대한 비교 연산을 수행함으로써 IME의 계산 복잡도를 줄이는 연구가 이루어졌다 [9-20]. 이러한 연구로는 three-step search (3SS), new three-step search (N3SS), diamond search (DS), four-step search (4SS), unrestricted center-biased diamond search (UCBDS), cross diamond search (CDS), the hexagon-based search (HEXBS) 등이 있다 [9-15]. 또한, 이러한 연구들을 기반으로 하여, 영상의 움직임 특성에 따라 적응적으로 다양한 탐색 패턴을 선택하여 사용하는 연구들이 이루어졌다 [16-20]. 이러한 적응적 탐색 패턴을 활용하는 대표적인 연구로는 test zone search (TZS)가 있는데, 이 알고리즘은 diamond 탐색 패턴과 raster 탐색 패턴을 적응적으로 활용하였다. 때문에 TZS 알고리즘은 움직임이 단순한 영상에 대해서는 빠르게 MV를 찾을 수 있고, 움직임이 복잡한 영상에 대해서는 정확한 MV를 찾을 수 있다. 더불어 TZS 알고리즘을 개선하기 위한 노력 또한 이루어져 TZS를 기반으로 하는 변형된 TZS가 다수 연구되었다 [21-24].

TZS는 HM 참조 소프트웨어의 IME 알고리즘으로 채택되어 널리 사용되고 있으며, 본 논문에서도 참조하여 사용하고 있기 때문에 TZS에 대하여 조금 더 자세히 설명한다. 그림 2.5에 HM 13.0에 사용된 TZS알고리즘을 블록 다이어그램으로 나타내었다. 가장 먼저 움직임 탐색을 시작하기 전에 MVP를 구한다. MVP가 다수인 경우 MVP 위치에 대해 비교 블록 매칭 연산을 수행하여 가장 유사한 블록 위치의 MVP를 하나 선택한다. 선택된 MVP로 초기 탐색 중심을 결정하고, 탐색 중심 주변의 영역을 탐색 범위로 정한다. HM 참조 소프트웨어에 사용된 TZS의 경우, AMVP를 활용하여 초기 탐색 중심을 결정한다. 탐색 중심 결정이 완료되면, 본격적인 움직임 탐색을 시작한다. 가장 먼저 탐색 중심 주변으로 diamond 탐색 패턴으로 블록 매칭을 수행하여 가장 유사한 블록 위치를 결정한다. Diamond 탐색 패턴은 그림 2.5에 우측 가운데에 나타나 있으며, 그림에서는 탐색 범위가 4인 diamond 탐색 패턴의 예를 나타내었다. 다음으로, 앞서 찾아진 최적 MV와 diamond 탐색 패턴의 중심점 사이의 거리가 ‘d’보다 더 큰 경우 MV가 클 가능성이 높은 것으로 판단하여 raster 탐색을 이어서 수행한다. 반면에, 초기 diamond 탐색 결과가 ‘d’ 이하의 거리를 가지는 경우 raster 탐색을 생략하고, 즉시 refinement 탐색 과정으로 진행하게 된다. Raster 탐색 패턴은 탐색 범위 전체를 가로방향 및 세로 방향으로 ‘d’ 간격으로 샘플된 위치를 탐색하도록 한다. 그림에서는 ‘d’가 3인 경우의 raster 탐색 패턴을 나타낸다. 다음으로 raster 탐색이 종료되거나, 초기 diamond 탐색 과정에서 ‘d ‘ 이하의 거리를 가진 경우 refinement 탐색 과정으로 진행하게 된다. Refinement 탐색 과정은 초기 diamond 탐색 패턴과 동일하지만, 종료 조건이 만족 될 때까지 diamond 탐색을 반복적으로 수행된다. 매 diamond 탐색을 수행할 때마다 찾아진 최적의 MV가

새로운 diamond 탐색의 탐색 중심이 되며, 최적 MV가 현재 diamond 탐색 패턴의 탐색 중심이 될 때 refinement 탐색을 중단한다. 이 조건은 'd'가 0이되는 것과 같다.

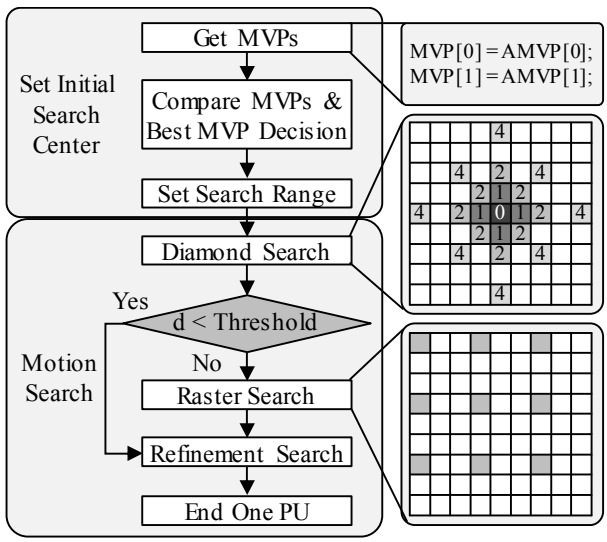


그림 2.5 HM 13.0 에 사용된 Test zone search (TZS) 알고리즘의 블록다이어그램

2.2.2 고속 Merge Mode Estimation 알고리즘

MME는 HEVC에 최초로 도입되어, 비교적 최근에 연구가 시작되었고, ME에 비해 계산 복잡도가 적기 때문에, MME에 대하여 다룬 연구는 상대적으로 그 수가 적다 [25], [26]. ME의 결과를 바탕으로 MME의 계산 복잡도를 낮추는 방법을 제안한 한 연구는, ME의 결과가 실제 MV를

가장 정확하게 나타내고 있으므로 ME가 찾아낸 실제 MV의 위치에 가까운 merge 후보만을 선택하여 MME를 수행하는 방법을 다루었다. 이 연구가 제안하는 방법은 ME의 수행 결과 최적의 참조 픽쳐와 MV가 결정되면, 이 최적의 참조 픽쳐와 동일한 참조 픽쳐를 가리키는 merge 후보만을 선택한다. 또, MVD 측면에서, 실제 MV와 merge 후보의 MV의 차이가 일정 threshold 이상이면, merge 후보가 실제 MV로부터 먼 위치를 가리키게 되므로 실제 MV가 아닐 가능성이 없다고 판단한다. 더불어 merge 후보들의 공간적, 시간적으로 인접한 위치에 따라 최적 merge 후보로 선택되는 확률을 통계로 구하여, 높은 빈도로 선택되는 후보를 우선적으로 확인 하도록 한다. 이 연구는 최종적으로 다섯 개의 merge 후보 중 하나의 후보만 선택하도록 함으로써, 계산 복잡도를 80% 감소시킬 수 있지만, 압축 효율의 저하가 2~3%의 BD-BR증가로 크게 나타나는 단점을 가지고 있다.

2.3 화면 간 예측 하드웨어 구조에 대한 이전 연구

2.3.1 하드웨어 기반 Integer Motion Estimation 연구

H.264/AVC가 발표된 이후부터 실시간 영상 인코딩 및 디코딩에 대한 수요가 급증하면서, 하드웨어 기반 IME에 대한 연구가 매우 활발하게 이루어졌다. IME에는 원본 블록과 참조 픽쳐내 블록 간의 차이를 측정하기 위해 sum of absolute difference (SAD) 연산이 사용된다. SAD 이외에도 sum of absolute transformed difference (SATD), sum of squared difference (SSD) 등의 다양한 연산이 존재하지만, SAD가 가장 간단하고

충분한 정확도를 가지는 것으로 알려져 IME를 위한 하드웨어 연구의 초기에는 SAD 연산을 빠르게 수행하기 위한 하드웨어 구조가 다수 연구되었다. 많은 연구들 중, 구조가 간단하고 빠른 SAD 연산 능력을 갖는 SAD-tree 구조가 오늘날에는 가장 널리 사용되고 있다 [27].

SAD-tree는 특정 블록의 SAD를 계산하기 위해, 블록 내의 모든 픽셀들이 갖는 원본 픽쳐와 참조 픽쳐 간의 차이 값의 절대값을 모두 더하는 과정을 트리 구조로 구현하였다. 예를 들면, 16x16 블록에 대해 SAD 연산을 수행하기 위해서는 총 256개의 차이값의 절대값을 더해야 하는데, 인접한 픽셀들의 차이값의 절대값을 4개씩 묶어 더하는 연산을 수행하는 모듈을 총 64개를 사용하여, 2x2 pixel에 대한 SAD를 구하고 다시 공간적으로 인접한 2x2 SAD 값을 4개를 묶어 4x4 SAD를 구하고 다시 이 4x4 SAD를 4개 묶어 8x8 SAD를 구성하고, 최종적으로 8x8 SAD 4개를 4개 묶어 16x16 SAD를 구하는 구조를 가진다. SAD-tree는 구조가 매우 직관적이고, 하드웨어의 특징인 병렬화 특성을 십분 활용할 수 있다는 점도 우수하지만, 이에 더하여, 트리의 각 계층의 중간 계산값을 활용하면, SAD-tree 크기의 SAD 값뿐 만 아니라 SAD-tree 내에 포함되는 작은 블록들에 대한 SAD 값을 동시에 구할 수 있다는 장점을 가진다.

하드웨어 기반 IME에서 빠른 SAD 연산도 중요하지만, 연산을 수행하기 위해 필요한 데이터를 제때 공급해 줄 수 있는 메모리 구조 또한 매우 중요하다. IME의 특징 중 하나인 참조 픽쳐 메모리에 대한 반복적이고 빈번한 접근이 하드웨어 기반 IME에서 병목현상을 일으키는 주요한 원인이 되기 때문에, IME의 참조 데이터 접근에 대한 연구 또한 활발하게 이루어졌다 [28-32]. 대부분의 연구에서 공통적으로 외부 메모리와 내부 메모리를 모두 활용하는 메모리 계층 구조를 사용하여, 반복적으로 접근되거나, 접근이 예상되는 참조 데이터를 내부 메모리에

저장하고, 그렇지 않은 참조 데이터를 외부 메모리에 저장하는 방식을 사용하였다. 이 때 내부 메모리에 어떤 데이터를 저장하고 어떤 방식으로 저장할 것인가에 대한 다양한 방식이 연구되었다. 이 연구들은 크게 두 가지로 분류 될 수 있다. 첫 번째는 현재 IME 중인 블록 주변을 중심으로 하여 탐색 영역에 해당하는 참조 픽쳐 픽셀을 내부 메모리에 저장하고, IME를 수행중인 블록이 바뀔 때 마다, 블록의 변경에 따라 추가로 필요한 참조 픽쳐 데이터만 읽어 들이는 방식을 사용하는 방식이 있다. 이 방식은 level C, level C+ reuse 방법 등으로 불리며, 이 방법은 정적인 탐색 범위를 갖는 IME 하드웨어에 주로 사용되었다. 두 번째는 보다 적응적인 고속 IME 알고리즘을 사용하기 위해서 캐시 구조를 활용하여 내부 메모리에 저장되어야 할 데이터를 관리하는 방식을 사용하였다. 이 방식은 주로 적응적인 탐색 범위를 갖는 IME 알고리즘에 사용되었다.

앞서 살펴본 것과 같이 많은 논문들이 하드웨어 기반의 IME를 위해 연산 엔진과 참조 데이터 접근 방법에 대해 다루어 왔지만, 최근의 하드웨어 기반의 IME 연구에는 고성능의 고속 IME 알고리즘을 사용하려는 새로운 시도가 나타나고 있다. 전통적으로 병렬 수행과 높은 사용률을 우선하는 하드웨어 기반의 IME에서는 고속 IME 알고리즘은 사용되지 않는 것이 일반적이었다. 이는 고속 IME 알고리즘은 상황에 따라 최적의 IME를 수행하기 위해 다양한 조건을 탐색 과정 중에 지속, 반복적으로 체크하여 탐색 전략에 반영하기 때문에 하드웨어 구현에 어려움이 많기 때문이다. 게다가, 고속 IME 알고리즘을 사용하여 계산 복잡도를 줄이지 않더라도 하드웨어의 병렬 처리를 통한 높은 연산 능력을 활용하면 실시간 동작에 충분한 IME 성능을 얻을 수 있었다. 그러나, 최근 초고해상도 영상에 대한 수요가 증가함에 따라 IME의

계산 복잡도가 급격하게 증가하였고, 이로 인해 단순한 하드웨어의 병렬 연산 능력의 강화만으로는 높은 IME 성능을 얻기 어려워졌다 [33]. 때문에, 최근의 연구들은 하드웨어 기반의 IME를 위해 고속 IME 알고리즘들을 변형, 단순화 하여 병렬화가 용이하고 알고리즘의 단계별 의존성이 적은 하드웨어 향 IME 알고리즘을 제안했다 [34-37]. 그러나, 여전히 이 고속 IME 알고리즘의 성능은 본래의 고속 IME 알고리즘의 성능에는 미치지 못한다. 왜냐하면, 고속 IME 알고리즘이 훌륭한 계산 복잡도 감소 성능을 가질 수 있도록 하는 주요 장치 중 하나인 단계별 의존성이 하드웨어 향 IME 알고리즘에서는 하드웨어 병렬화 및 파이프라인을 저해하는 원인으로 지목되어 제거되었기 때문이다.

또한, 고속 IME 알고리즘이 하드웨어에 적합하지 않는 이유 중 다른 하나는, 참조 메모리에 대한 접근 문제 때문이다. 앞서 살펴본 바와 같이 과거의 많은 연구에서 하드웨어 기반의 IME를 위해 참조 데이터를 시기 적절하게 공급하기 위한 다양한 메모리 계층 구조가 제안되었다. 그러나, 이러한 노력에도 불구하고 여전히 하드웨어 기반의 IME에서는 고속 IME 알고리즘이 필요로 하는 참조 데이터접근을 시기 적절하게 제공하지 못한다. 첫 번째 원인은 고속 IME 알고리즘은 각 단계 별 결과에 따라서 다음 단계에서 사용될 참조 데이터가 변화하기 때문이다. 이로 인해, 필요한 참조 데이터가 규명된 시점으로부터 데이터의 fetch가 이루어질 때까지 시간만큼의 지연이 불가피하다. 두 번째 원인은 고속 IME 알고리즘이 필요로 하는 참조 데이터의 형태와 양이 불규칙적이기 때문이다. 일정한 형태의 접근만 허용하는 메모리의 태생적 특성은 원하는 참조 데이터를 fetch 하기 위해 여러 차례의 메모리 접근을 발생킨다. 때문에 고속 IME 알고리즘은 뛰어난 계산 복잡도 감소 성능과 준수한 압축 효율을 제공함에도 불구하고 참조 메모리 접근

문제에 의해 완전한 형태의 도입에 어려움을 겪고있다

2.3.2 하드웨어 기반 Merge Mode Estimation 연구

하드웨어 기반의 MME에 관한 연구를 중점적으로 다루고 있는 발표된 연구는 많지 않다. 그러나, 몇몇 연구를 통해 MME가 갖는 참조 데이터 접근 특성이 하드웨어에서 IME와 유사한 참조 데이터 접근 문제를 일으키고 있음이 확인되었다 [38]. 이 연구는 하드웨어 기반 MME에서 MME 후보의 MV가 빠르게 fetch가 불가능한 영역을 가리키고 있는 경우 예측을 생략하는 방법을 제안하였다. 그러나, MME의 연산에 대한 직접적인 연구는 거의 이루어지지 않았다. 왜냐하면, 현존하는 대부분의 하드웨어 기반의 inter prediction은 HEVC에서 새롭게 채택된 MME를 고려하지 않은 과거 H.264/AVC 또는 이전의 영상 압축 표준의 설계 방법이 적용된 것으로 단지 ME를 위해서만 설계되었기 때문이다. 이러한 하드웨어 기반의 inter prediction에서 MME는 탐색 지점이 제한된 ME 처럼 취급되어왔으며, 단지 ME를 위한 하드웨어 리소스를 MME를 위해서도 사용하는 방식을 사용하여 MME를 수행할 수 있기 때문에, MME를 위한 새로운 하드웨어 구조의 필요성이 적었기 때문이다. 그러나, 이러한 방식으로 구현된 MME는 기존 ME 하드웨어의 동작을 방해하거나, 방해 받을 수 있다. 또한 MME는 이웃하는 PU에 대한 의존성으로 인해 가장 마지막 파이프라인 스테이지에서 이루어져야 하므로, ME와 MME는 서로 다른 파이프라인 스테이지에서 동작하여, 참조 데이터 접근의 locality를 떨어트리는 요인이 될 수 있다 [36]. 그러므로, 보다 효율적인 MME의 수행을 위해서는 MME 전용의 효율적인 하드웨어 구조가 필수적이다.

제 3 장 Bottom-up Integer Motion Estimation

3.1 서로 다른 계층 간의 Motion Vector 관계 관찰

3.1.1 서로 다른 계층 간의 Motion Vector 관계 분석

이 절에서는 서로 다른 depth에 속하는 PU들이 갖는 움직임 정보의 관계에 대한 실험 결과를 보인다. 이 절에서 보이는 실험들은 움직임 정보를 다른 depth의 PU로부터 예측하기 위한 방법을 찾기 위한 것 이다. 그림 3.1에 depth d와 depth d+1의 두개의 서로 다른 depth에 대한 inter prediction의 예를 나타내었다. 그림 3.1 (a)에 나타낸 depth d 의 CU는 한 개의 2Nx2N PU를 가지며, depth d+1의 CU들은 다섯 개의 PU를 갖는다. 이 때 다섯 개의 PU들은 모두 depth d 의 2Nx2N PU의 픽셀 영역에 포함된다. 그림 3.1 (b)에 나타낸 depth d의 CU는 두 개의 PU를 가진다. 첫 번째 nLx2N PU는 depth d+1의 다섯 개의 PU들과 픽셀 영역을 공유하며, 두 번째 nLx2N PU는 depth d+1의 세 개의 PU들과 픽셀 영역을 공유한다. 두 개의 depth는 HEVC의 블록 분할 구조에 따라 서로 다른 블록 분할 구조를 가질 수 있지만, 같은 픽셀 영역을 공유하는 특성을 가지게 된다. 그러므로, 그림 3.1에 나타낸 것과 같이 두 depth의 PU들은 매우 유사한 MV를 가지게 된다. 예를 들어 그림 3.1 (a)의 두 가지 depth의 모든 PU들은 (7,3) 또는 (7,4)의 MV 값을 갖는다. 그림 3.1 (b)의 MV는 두 가지 종류의 MV 로 나뉜다. 한 가지는 (1,2)와 (2,3)의 MV를

갖는 그룹이며, 다른 한 가지는 (9,7)과 (9,8)의 MV를 갖는 그룹으로 나뉜다.

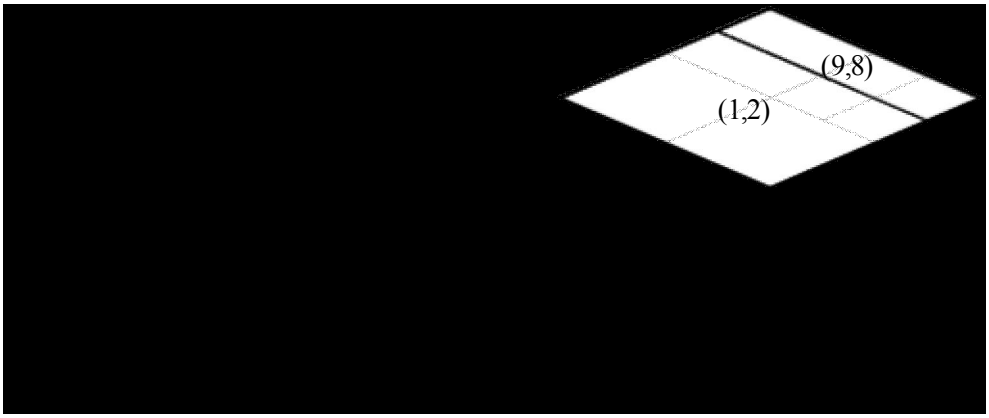


그림 3.1 서로 다른 2 개의 depth d, depth d+1 에 대한 inter prediction의 예

그림 3.1에 나타낸 서로 다른 depth에서 발생하는 HEVC의 움직임 정보의 유사성에 대해서 정량적인 관찰을 수행하기 위해, HEVC의 사용 가능한 모든 depth 간의 움직임 벡터에 대한 통계적 관계를 시뮬레이션을 통해 확인하였다. 표 3.1과 표 3.2에 두개의 서로 다른 depth간의 MV 차이의 평균값과 표준 편차를 각각 나타내었다. 총 16개의 비디오가 LDP 실험 환경으로 테스트 되었으며, ME를 완전히 마친 후 4x4 pixel 블록 단위로 통계값이 수집되었다. 또한 통계값의 단위는 맨하탄 거리 단위를 사용하였다. 표는 각 테스트 영상 별로, HEVC에서 발생할 수 있는 모든 두 개의 depth간의 관계에 대한 통계값을 나타낸다. 예를 들어, Traffic 영상의 depth 2와 depth 3 사이의

MV 차이값의 평균은 0.54 픽셀이며 표준 편차는 2.36 픽셀이다. 또 PartyScene 영상의 depth 0과 depth 3 사이의 MV 차이값의 평균은 1.52 픽셀이며 표준 편차는 6.05 픽셀이다. 각 표에서 진한 회색으로 음영 처리된 부분이 인접한 depth간의 통계값을 나타내고, 연한 회색으로 음영 처리된 부분이 이웃하지 않은 depth간의 통계값을 나타낸다. 시뮬레이션 결과는 모든 실험 영상에 대해서, 어떤 두 depth관의 관계라고 하더라도 MV 차이값의 평균은 약 6 픽셀 이하이며, 서로 인접한 depth의 경우 그 차이값은 다섯 픽셀로 더욱 작음을 보여준다. 반면, 표준 편차의 경우 depth간의 거리보다는 영상 자체의 움직임 복잡도에 더욱 영향을 많이 받는다. 예를 들어, Traffic, FourPeople, Johnny와 같이 매우 느리고 단순한 움직임을 갖는 영상의 경우 표준편차는 depth의 거리가 가까울수록 작아지는 경향을 보이지만, PeopleOnStreet, BasketballDrive, RaceHorses 등과 같이 빠르고 복잡한 움직임을 갖는 영상의 경우 이 경향성은 약화되어 depth간의 거리에 영향을 덜 받게 된다. 표. 3.1와 표 3.2를 통해 관찰된 서로 다른 depth간의 거리와 MV 차이값의 관계는 대다수의 영상에서 매우 안정적이고 강한 관계를 보여준다.

표 3.1 서로 다른 depth 사이의 MV 차이값의 평균

	Average Difference of MVs between depths (pixel)											
	Traffic			PeopleOnStreet			BasketballDrive			BQTerrace		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	0.64	0.78	0.80	2.45	2.93	2.79	5.03	5.72	5.61	1.79	2.11	2.17
1	-	0.63	0.70	-	2.69	2.80	-	5.08	5.55	-	1.80	2.04
2	-	-	0.54	-	-	2.32	-	-	3.96	-	-	1.29
	Cactus			Kimono			ParkScene			BQMall		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	1.99	2.27	2.17	1.96	2.69	2.69	1.14	1.33	1.34	1.39	1.65	1.56
1	-	2.04	2.22	-	2.47	2.71	-	1.17	1.31	-	1.54	1.59
2	-	-	1.68	-	-	2.32	-	-	0.96	-	-	1.28
	BasketballDrill			PartyScene			RaceHorses			BasketballPass		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	2.28	2.54	2.45	1.21	1.56	1.52	4.01	4.65	4.42	1.05	1.28	1.29
1	-	2.19	2.31	-	1.43	1.52	-	4.26	4.45	-	1.21	1.28
2	-	-	1.76	-	-	1.38	-	-	3.51	-	-	0.90
	BlowingBubbles			RaceHorsesD			FourPeople			Johnny		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	1.54	1.86	1.69	2.17	2.68	2.63	0.37	0.40	0.38	0.66	0.71	0.69
1	-	1.71	1.78	-	2.64	2.73	-	0.31	0.33	-	0.42	0.45
2	-	-	1.54	-	-	2.17	-	-	0.22	-	-	0.30

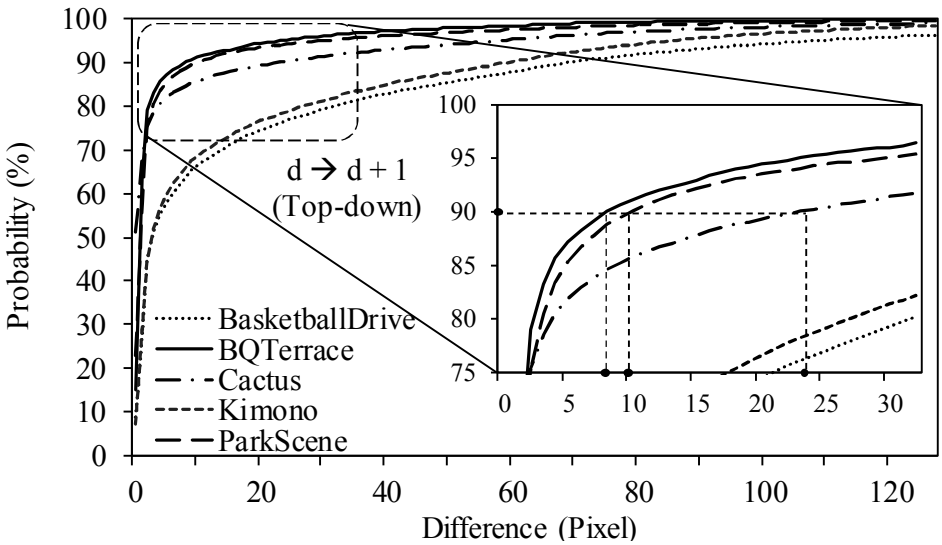
표 3.2 서로 다른 depth 사이의 MV 차이값의 표준편차

	Standard Deviation of MV between depths (pixel)											
	Traffic			PeopleOnStreet			BasketballDrive			BQTerrace		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	2.53	2.73	2.60	6.91	7.57	6.52	14.12	14.55	13.81	7.26	7.19	6.83
1	-	2.62	2.60	-	8.60	8.04	-	14.16	14.14	-	6.76	6.64
2	-	-	2.36	-	-	8.00	-	-	12.00	-	-	4.80
	Cactus			Kimono			ParkScene			BQMall		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	9.31	9.69	9.14	5.15	6.52	5.56	5.25	5.47	5.11	4.98	5.53	4.99
1	-	9.62	9.72	-	6.98	6.39	-	5.44	5.35	-	6.03	5.77
2	-	-	8.39	-	-	6.98	-	-	4.73	-	-	5.52
	BasketballDrill			PartyScene			RaceHorses			BasketballPass		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	8.17	8.30	7.55	6.02	6.77	6.05	10.91	11.31	9.99	3.43	3.76	3.35
1	-	8.57	8.21	-	7.13	6.77	-	11.94	11.32	-	4.37	4.06
2	-	-	7.31	-	-	6.80	-	-	10.38	-	-	3.71
	BlowingBubbles			RaceHorsesD			FourPeople			Johnny		
Depth	1	2	3	1	2	3	1	2	3	1	2	3
0	7.99	8.34	7.20	6.19	6.88	5.91	2.02	2.16	1.99	2.57	2.54	2.48
1	-	8.67	8.34	-	7.99	7.38	-	2.07	2.04	-	1.54	1.51
2	-	-	7.93	-	-	7.26	-	-	1.82	-	-	1.19

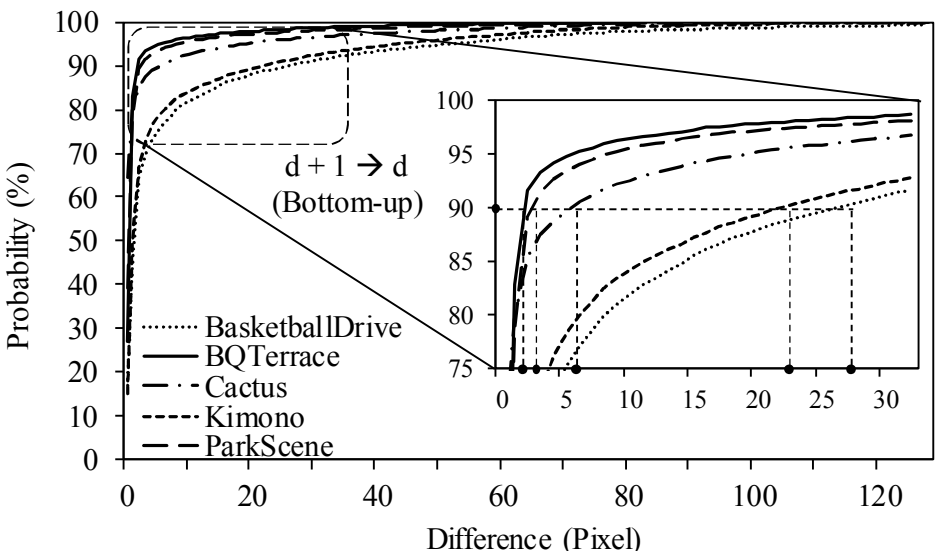
3.1.2 Top-down 및 Bottom-up 방향의 Motion Vector 관계 분석

이 절에서는 앞서 살펴본 서로 다른 depth간의 MV 관계를 다른 depth 의 MV로부터 현재 depth의 MV를 예측 하는 데에 활용하기 위해 다른 실험 결과를 보인다. 그림 3.2는 클래스 B의 다섯 개의 영상에 대한 인접한 depth 간의 MV 차이값의 누적 분포 함수를 나타낸다. 차이값은 맨하탄 거리로 측정 되었다. 그래프의 가로축은 서로 이웃한 depth 간의 MV 차이값의 크기를 나타내며, 세로축은 누적 분포 확률을 나타낸다. 그림 3.2 (a)는 depth d의 MV가 depth d+1에 주어진 MV 차이값으로 나타날 확률을 나타내며, 반면에 그림 3.2 (b)는 depth d+1의 MV가 depth d에 주어진 MV의 차이값으로 나타날 확률을 나타낸다. 즉, 그림 3.2 (a)는 상위 depth의 MV로 하위 depth 의 MV을 예측 할 때의 정확도를 나타내며, 그림 3.2 (b)는 하위 depth의 MV로 상위 depth의 MV를 예측 할 때의 정확도를 나타낸 다고 할 수 있다. 본 논문에서는 그림 3.2 (a)의 MV 추측 방향을 top-down 이라고 부르며 그림 3.2 (b)의 MV 추측 방향을 bottom-up이라고 부르기로 한다. 그림 3.2에 나타난 것과 같이 현재 depth의 MV는 이웃 depth의 MV와 높은 확률로 가까운 거리에서 발견되는 것을 볼 수 있다. 특히, 가까운 거리에서 MV가 발견될 확률은 depth d+1로부터 depth d 보다는 depth d 로부터 depth d+1를 관찰할 때에 더욱 높다. 이는 매우 자연스러운 것으로, 앞서 그림 3.1에 나타낸 것과 같이, depth d+1에 속하는 모든 PU들은 depth d의 PU들에 대부분 완전히 포함되기 때문이다. 이러한 포함 관계로 인하여, 동일한 픽셀 영역을 갖는 depth d+1의 다수의 PU들의 MV는 depth d 의 PU의 MV를 추정하는 데에 사용될 수 있다. 이와는 반대로 depth d+1의 많은 수의 PU에 대해서 depth d의 단일 PU로 MV를 추정하는 것은

어렵다. 그러므로 depth d+1로부터 depth d의 MV를 추정하는 것이 더욱 정확하다. 이는 그림 3.2에도 잘 나타나 있다. 그림 3.2 (a)에서 움직임이 단순한 BQTerrace영상의 경우 depth d의 MV의 83% 이상이 depth d+1의 MV들과 하나의 픽셀 거리 이내의 차이만을 가진다. 움직임이 단순한 다른 영상 ParkScene에 대해서는 90% 이상의 depth d의 MV가 depth d+1의 MV와 세 픽셀 거리 이내의 차이만을 갖는다. 움직임이 복잡한 BasketballDrive와 Kimono에 대해서는 두 depth의 MV의 차이값이 커지는 경향을 보이지만 여전히 top-down 보다 bottom-up 방향으로 MV를 추정하였을 때 더욱 높은 정확도를 보인다.



(a)



(b)

그림 3.2 인접한 depth 간의 MV 차이값의 누적 분포 함수: (a) Top-down 방향, (b) Bottom-up 방향

3.2 Bottom-up Motion Vector Prediction

CTU를 인코딩 하기 위해서 예측을 수행하는 순서는 큰 CU로부터 작은 CU의 순서 또는 작은 CU로부터 큰 CU로 어느 쪽도 선택이 가능하다. 본 논문에서는 전자를 top-down 순서라고 지칭하고, 후자를 bottom-up 순서라고 지칭한다. 3.1절에서 살펴본 것과 같이 bottom-up 순서가 MV를 더욱 높은 정확도로 추정할 수 있으므로 본 논문에서는 정확한 MV 예측을 위해 bottom-up 순서를 사용한다. 기존의 HEVC 표준에서 사용되는 MVP는 AMVP라고 지칭하며, 본 논문에서 제안하는 bottom-up 순서를 통해 유도되는 MVP를 bottom-up MVP (BMVP) 라고 지칭한다. 주의할 점은 BMVP는 오직 IME의 탐색 중심을 결정하는 데에만 사용되며 비트스트림을 생성하기 위해서는, 기존의 AMVP가 사용된다. 즉 제안된 MV 예측 방법은 HEVC 표준을 완전히 따른다.

그림 3.3은 제안하는 MV 예측 방법의 순서도를 나타낸다. 이 과정은 매 참조 픽쳐 마다 수행되어 IME의 MV 탐색 이전에 탐색 중심을 결정한다. 먼저, 현재 CU가 가장 하위 depth에 속하여 가장 작은 CU 크기인 경우 현재 PU를 위해 두 개의 AMVP가 유도된다. 현재 CU가 가장 작은 CU가 아닌 경우 AMVP와 BMVP가 모두 사용된다. 첫번째로, 현재 CU와 동일한 픽셀 영역을 갖는 하위 depth의 PU들을 확인한다. 이러한 PU들을 subPU라고 지칭한다. 즉, 현재 CU의 depth가 d 라고 하면 subPU는 depth d+1에 속하게 된다. HEVC의 쿼드-트리 블록 분할 구조에 따라 subPU의 수는 최소 2개에서 최대 8개가 될 수 있다. 예를 들어 그림 3.1 (a)의 경우 depth d의 PU는 다섯 개의 subPU를 가지며, 그림 3.1 (b)의 경우 depth d의 두 개의 PU는 각각 다섯 개와 세 개의 subPU를 갖는다. 다음으로 subPU의 모든 MV들이 현재 PU를 위한 MVP

리스트에 추가된다. 이후 이 MVP 리스트에는 2개의 AMVP 또한 추가된다. 이렇게 구성된 MVP 리스트 내에서 중복되는 MVP는 제거된다. 그림 3.1 (a)의 예에서, depth d의 2Nx2N PU는 2개의 BMVP를 갖는다. 이 BMVP는 (7,3), (7,4)이다. 또한 그림 3.1 (b)의 예에서 다섯 개의 subPU를 갖는 첫 번째 PU는세 개의 BMVP를 갖는다. 이 BMVP는 (1,2), (1,3), (9,7)이다. 이와 마찬가지로 세 개의 subPU를 갖는 두 번째 PU는 세 개의 BMVP를 갖는다. 이 BMVP는 (9,8), (9,7), (1,3)이다. 2개의 AMVP를 추가로 포함하면 최대 10개의 MVP가 MVP 리스트에 포함될 수 있다.

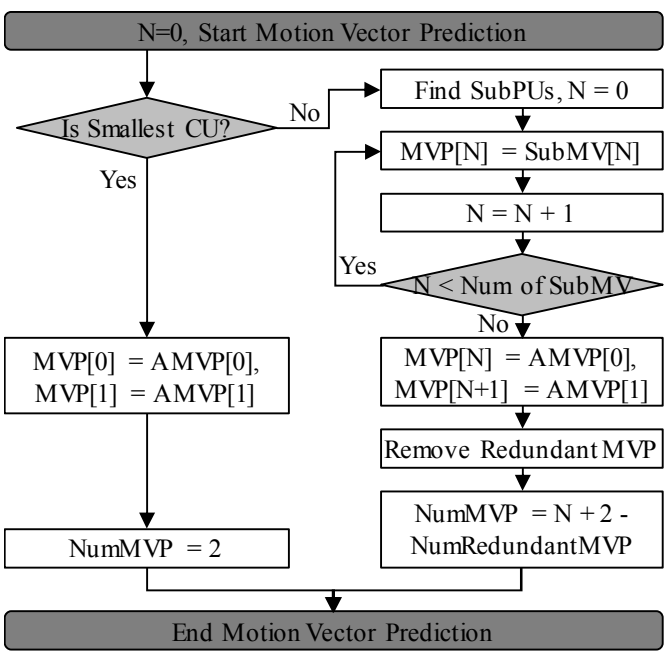


그림 3.3 제안하는 Bottom-up motion vector prediction의 순서도

BMVP는 언제나 바로 인접한 하위 depth로부터 매 depth 별로 유도된다. 이러한 계층별 단계적 BMVP의 유도는, 인접하지 않은 하위 depth로부터 BMVP를 유도하는 것 보다 정확하다. 또한 각 depth에서 BMVP를 활용한 ME는 다음 상위 depth를 위한 오차 보상 기능을 갖기 때문에 바로 인접한 depth 로부터 BMVP를 유도하는 방법이 가장 정확한 MVP를 유도할 수 있는 방법이 된다.

그림 3.4에 제안하는 MV 예측 방법과 HM의 AMVP를 사용한 MV 예측 방법을 유도된 MVP의 수와 MVP와 실제 MV사이의 거리를 다양한 영상에 대해 나타내었다. LDP 실험 조건이 사용되었다. 가로축은 여러가지 영상을 나타내고 좌측 세로축은 실제 MV와 MVP간의 거리의 평균값을 나타내고, 우측 세로축은 유도된 MVP의 수를 나타낸다. 연한 회색 및 진한 회색 그래프는 AMVP와 제안된 BMVP 예측 방법의 평균 MV 거리를 나타낸다. 마름모 및 동그라미는 각각의 MV 예측 방법에 따라 유도된 MVP의 평균 수를 나타낸다. BMVP는 8x8 CU에 대해서는 사용되지 않기 때문에 오직 16x16, 32x32, 64x64 CU에 대해서만 보인다. 움직임이 복잡하고 빠른 PeopleOnStreet, BasketballDrive, Kimono, RaceHorses 영상에 대해서 MVP의 수는 상대적으로 많다. 반면에 움직임이 정적인 Traffic, FourPeople, Johnny, KristenAndSara, 영상에 대해서는 MVP의 수는 상대적으로 적다. BMVP는 쿼드-트리 구조에 따라 최대 8개까지 유도 될 수 있으나, 평균 약 4개가 유도된다. 많은 수의 MVP는 정확한 MV를 예측하는 데에 도움이 될 수 있지만, 오히려 정확하지 못한 MVP들은 ME과정에 오히려 계산 복잡도를 증가시키는 결과를 야기할 수 있다. 그러므로 유도된 MVP와 실제 MV 사이의 거리 또한 그림 3.4에서 관찰 되었다. 모든 영상에 대해서 제안된 BMVP는 AMVP보다 평균 약 14% 의 MVP의 거리 오차를 감소시킨 것을 볼 수

있다. 평균 14%의 거리 오차 감소는 적은 것으로 평가될 수 있으나, 14%의 오차 감소는 IME의 계산 복잡도를 큰 폭으로 개선 시킨다. 이는 다수의 MVP가 포함된 MVP 리스트 중 적어도 하나의 정확한 MVP가 존재하면 고속 IME 알고리즘은 매우 빠르게 실제 MV에 수렴할 수 있기 때문이다.

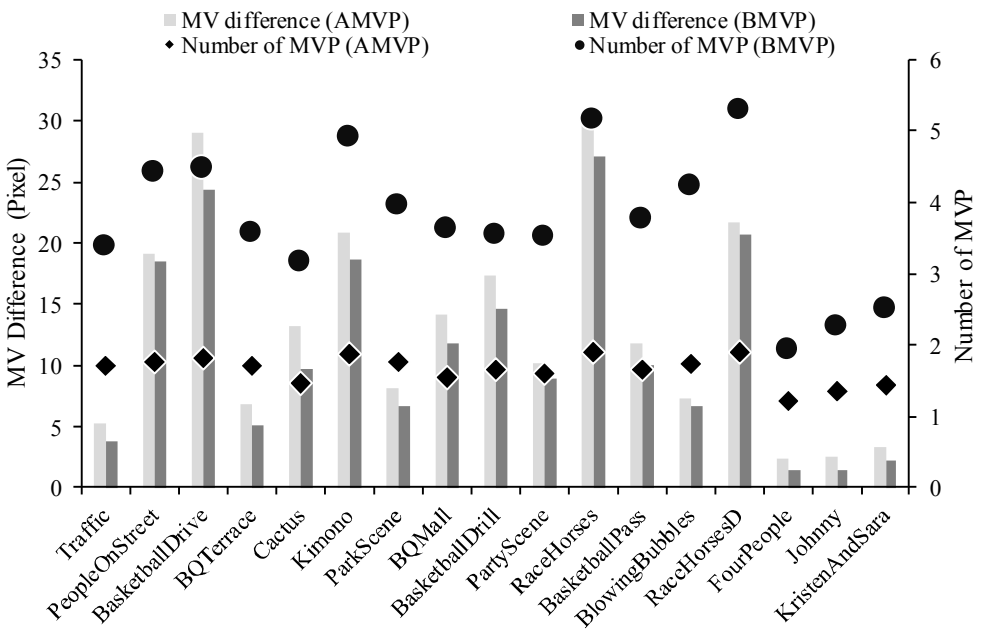


그림 3.4 AMVP와 제안하는 MV 예측 방법으로 유도된 MVP의 실제 MV와의 평균 거리와 유도된 평균 MVP의 수

3.3 Bottom-up Integer Motion Estimation

3.3.1 Bottom-up Integer Motion Estimation - Single MVP

이 절에서는 3.2절에서 제안한 MV 예측 방법을 기반으로 한 고속 IME 알고리즘을 소개한다. 그림 3.5에 제안하는 고속 IME 알고리즘을 나타내었다. 제안하는 고속 IME 알고리즘은 MV 예측, 탐색 범위 결정 및 움직임 탐색의 세 가지 단계로 이루어져 있다. 첫 단계인 MV 예측은 앞서 제안한 MV 예측 방법으로 이루어진다. 두 번째 단계는 subPU가 존재하는 CU인지 아닌지에 따라 결정된다. 만약 현재 IME를 수행하는 PU가 가장 작은 CU에 속한다면 BMVP가 사용 가능하지 않으므로 넓은 탐색 범위를 지정한다. 반면에 높은 정확도를 갖는 BMVP를 사용할 수 있는 depth의 CU인 경우 좁은 탐색 범위를 지정한다. 다음으로 움직임 탐색 단계에서는 제안된 MV 예측 방법으로 구성된 MVP 리스트를 사용하여 움직임 탐색을 수행한다. 만약 다수의 MVP가 MVP 리스트에 포함되어 있다면, 움직임 탐색 과정은 다수의 MVP 위치에서 블록 매칭을 수행한 다음 가장 작은 SAD값을 갖는 위치의 MVP를 탐색 중심으로 하여 수행한다. 움직임 탐색을 위한 알고리즘으로는 어떠한 알고리즘도 사용 될 수 있지만, 본 논문에서는 HM 참조 소프트웨어에 채택되어 널리 사용되는 TZS를 사용하였다.

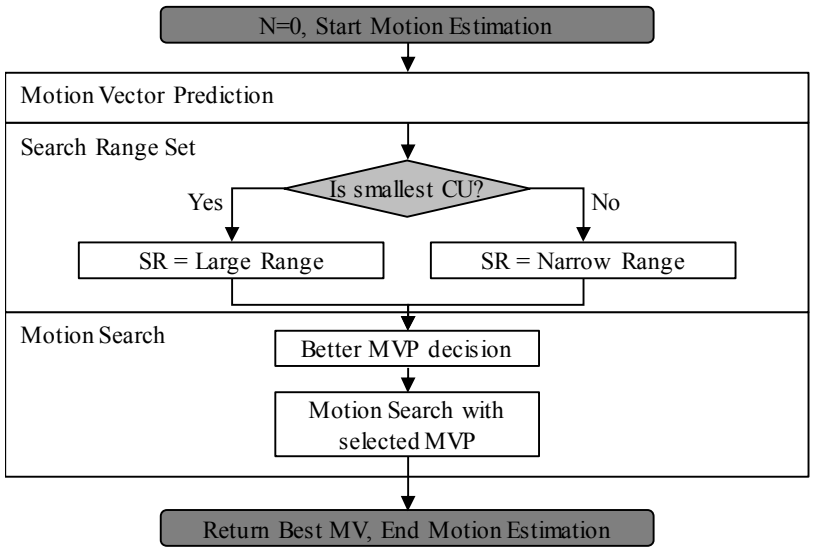


그림 3.5 제안된 MV 예측 방법을 적용한 단일 MVP기반의 Integer motion estimation의 순서도

3.3.2 Bottom-up Integer Motion Estimation – Multiple MVP

이 절에서는 3.2절에서 제안한 MV 예측 방법을 기반으로 다수의 MVP를 모두 활용하는 방법에 대해 설명한다. 3.2절에 설명한 것과 같이 BMVP는 AMVP에 비해 정확하고 그 수도 더 많다. 그러므로 높은 정확도를 갖는 다수의 MVP 중 하나만을 탐색 중심으로 결정하여 탐색을 수행하는 것은 더 빠르게 최적 MV를 찾을 수 있는 기회를 잃어버릴 가능성이 있다. 그러므로 다수의 MVP 모두를 활용하여 IME를 수행하는 방법을 그림 3.6과 같이 제안한다. 제안된 IME는 단일 MVP기반의 IME와 매우 유사하지만 움직임 탐색 단계에서 MVP를 한 개 선택하는 대신 다수의 MVP를 각각의 탐색 중심으로 설정하고 움직임 탐색을 MVP의 수만큼 반복하여 탐색하도록 하였다. 다수의

MVP를 탐색 중심으로 활용한 IME는 그 계산 복잡도가 단일 MVP를 탐색 중심으로 활용한 경우보다 MVP의 수만큼 증가의 우려가 있지만, 3.4에서 후술될 실험 결과 정확한 다수의 MVP를 사용하여 탐색을 진행하므로 더욱 작은 탐색 범위를 사용할 수 있기 때문에 단일 MVP의 경우보다 계산 복잡도 – 압축 효율 간의 트레이드 오프 관계가 더욱 효율적인 사실이 확인되었다. 그러므로 보다 좁은 복수의 탐색 범위에서도 뛰어난 성능의 압축 효율을 보여, 4장에서 후술 될 하드웨어 기반의 Bottom-up IME에 보다 적합하다.

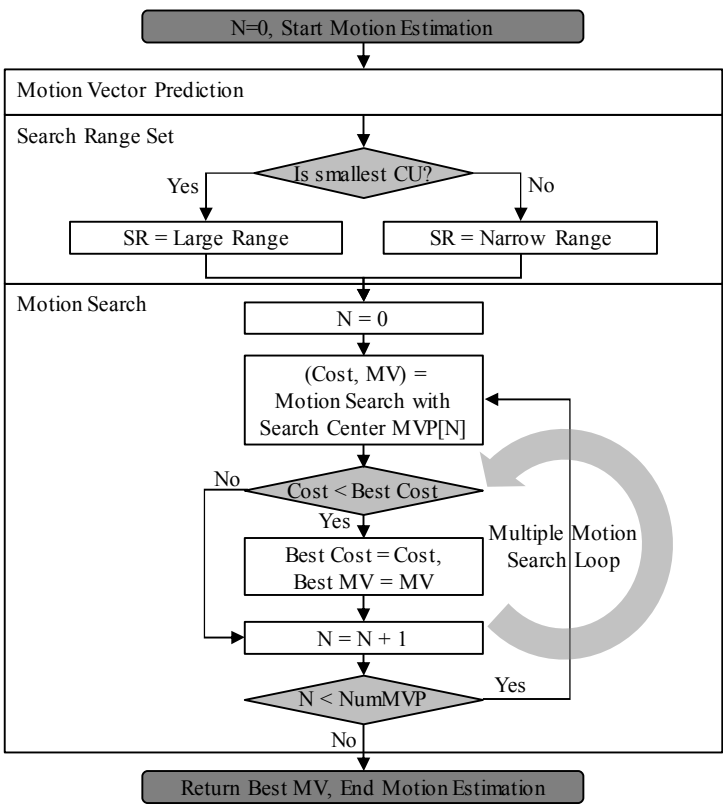


그림 3.6 제안된 MV 예측 방법을 적용한 복수 MVP기반의 Integer motion estimation의 순서도

3.4 실험 결과

이 절에서는 앞서 소개한 MV 예측 방법과 고속 IME 알고리즘을 적용하였을 때의 계산 복잡도 감소 성능 및 압축 효율의 변화를 보인다. 더불어 제안된 MV 예측 방법의 정확도를 평가하기 위한 실험이 수행되었다. 실험 환경은 1.3 절에 나타낸 것과 동일하며, 단지 사용되는 영상의 종류 및 추가 실험 QP 대역의 상이한 점이 존재한다. 상이한 점이 있는 실험의 경우 해당 실험의 설명에 상이한 부분을 명시하였다. IME의 계산 복잡도의 비교를 위하여 IME의 계산 복잡도를 식 (3.1)과 같이 구하였고, 계산 복잡도 변화량을 식 (3.2)와 같이 정의하였다. N_{pu}는 IME가 수행된 모든 PU의 수를 나타내며 NumPelPU는 각 PU를 구성하는 픽셀의 수를 나타낸다.

$$C = \sum_{i=0}^{N_{pu}} (\mathrm{NumSearchPoint}_i \times \mathrm{NumPelPU}_i) \tag{3.1}$$

$$\Delta C = \frac{C_{proposed} - C_{reference}}{C_{reference}} \times 100\% \tag{3.2}$$

가장 먼저, 제안된 MV 예측 방법을 사용하였을 때 MVP가 얼마나 실제 MV를 정확하게 예측 하였는지에 대해 평가해 본다. 표 3.3에는 기존 MV 예측 방법인 AMVP와, 제안된 MV 예측 방법을 각각 사용하였을 때, 최적의 MV를 찾기 위해서 필요한 탐색 범위를 최적 MV를 발견할 확률에 따라서 나타내었다. 표의 세번째 행이 최적의 MV를 찾을 확률을 나타내며 다른 행들의 숫자는 각 열의 확률로 MV를

찾기 위해서 필요한 탐색 범위를 나타내었다. 세 번째 열부터 여덟 번째 열 까지는 기존의 AMVP를 사용한 경우를 나타내며, 아홉 번째 열부터 열 네번째 열 까지는 제안된 Bottom-up MVP를 사용한 경우를 나타낸다. 예를 들어 PeopleOnStreet 영상에 대해서는 AMVP를 탐색 중심으로 하였을 때는 41 픽셀, 제안된 BMVP를 추가로 사용한 경우 12 픽셀 이내의 탐색 범위에서 90%의 확률로 최적 MV가 발견된다. 움직임이 매우 정적인 Traffic, FourPeople, Johnny, KristenAndSara 영상에 대해서는 상대적으로 매우 좁은 탐색 범위에서 높은 확률로 최적 MV가 발견된다. 그러나, 움직임이 복잡한 BasketballDrive와 RaceHorses 영상에 대해서는 AMVP의 경우 90%의 최적 MV를 찾기위해서는 두 영상 모두 56픽셀의 탐색 범위를 탐색해야 하지만, 제안된 MVP를 사용하면 단지 17 픽셀과 19 픽셀의 탐색 범위를 탐색하는 것 만으로 충분하다. 결과적으로 기존 AMVP를 사용한 경우 95%의 최적 MV를 찾기 위해서는 모든 영상에 대해 평균 36 픽셀의 탐색 범위를 탐색해야 하지만, 제안된 MVP를 사용한 경우 단지 평균 14 픽셀의 탐색 범위 만을 탐색해도 충분하다. 그러므로 제안된 MV 예측은 예측의 정확도를 향상시켜, 기존 AMVP 기반의 MV 예측 방법 대비 61.1 %의 탐색 범위를 감소 시켰다.

표 3.3 기존 MV 예측 방법과 제안된 방법을 사용하였을 때 최적의 MV를 찾을 확률 별 필요 탐색 범위

Video Sequence		Search Range for Probability of Finding the Best MV (pixel)											
		AMVP						Proposed					
		0.5	0.8	0.9	0.95	0.98	0.99	0.5	0.8	0.9	0.95	0.98	0.99
ClassA	Traffic	1	2	3	6	25	47	0	1	1	1	4	11
	PerpleOnStreet	2	17	41	57	64	64	1	4	12	26	47	57
ClassB	BasketballDrive	4	31	56	64	64	64	1	6	17	32	53	61
	BQTerrace	1	2	6	15	36	51	1	1	1	3	13	25
	Cactus	0	3	18	45	63	64	0	1	4	12	29	44
	Kimono1	3	20	42	57	64	64	1	3	11	24	42	53
	ParkScene	1	3	7	22	52	62	0	1	2	6	19	34
ClassC	BQMall	1	6	23	49	63	64	0	1	6	16	35	49
	BasketballDrill	1	6	34	55	63	64	0	1	7	20	40	53
	PartyScene	1	2	10	39	59	63	0	1	2	10	27	42
	RaceHorses	6	36	56	63	64	64	1	8	19	34	52	60
ClassD	BasketballPass	1	8	25	46	60	64	0	2	5	16	33	45
	BlowingBubbles	1	3	6	12	44	58	1	1	1	3	13	26
	RaceHorses	4	21	46	60	64	64	1	4	14	29	48	57
ClassE	FourPeople	0	1	1	3	8	17	0	0	1	1	3	7
	Johnny	0	1	3	5	9	15	0	1	1	2	5	8
	KristenAndSara	0	2	4	7	13	22	0	1	2	2	6	10
	Average	2	10	22	36	48	54	0	2	6	14	28	38

그림 3.7에는 3.3.1절에 제안된 MV 예측 방법을 적용한 단일 MVP 기반의 IME의 계산 복잡도 감소와 압축 효율의 변화를 나타내었다. 탐색 범위를 64 픽셀 로부터 0 픽셀로 감소시키며 계산 복잡도와 압축 효율의 변화를 관찰하였다. 0 픽셀의 탐색 범위는 단지 MVP위치에 대해서만 블록 매칭을 수행하는 것이다. 유의할 점은 이 실험을 위해서는 단지 제안된 MV 예측 방법만이 적용되었고, 다수의 MVP에 대해 반복적으로 움직임 탐색을 수행하지는 않았다. 그러므로 다수의 MVP 중 가장 작은 SAD 값을 갖는 위치의 MVP가 한 개만 선택 되어 움직임 탐색이 수행된 실험 결과이다. LDP 실험 조건이 사용하여 실험 영상을 인코딩 하였으며 각 클래스 별 평균 값을 나타내었다. 막대

그래프는 압축 효율의 변화를 나타내고, 꺾은선 그래프는 계산 복잡도의 변화를 나타낸다. 가로축은 상위 depth의 CU를 위해 사용된 탐색 범위를 나타낸다. 최하위 depth인 8x8 CU에 대해서는 64픽셀의 탐색 범위가 적용되었다. 64 픽셀의 탐색 범위가 사용된 경우 평균 0.05%의 BD-BR 감소를 보여 압축 효율이 약간 개선되면서도 41%의 IME 복잡도가 감소 성능을 얻었다. 탐색 범위를 64 픽셀 로부터 8픽셀까지 감소시키면 계산 복잡도는 꾸준하게 감소하지만, 압축 효율의 유의미한 감소는 나타나지 않는다. 이는 표 3.3에 나타낸 것과 같이 정확한 MV 예측으로 인해 좁은 범위의 탐색 범위를 사용하더라도 최적 MV를 높은 확률로 찾을 수 있기 때문이다. 탐색 범위가 4 픽셀 이하로 작아지면 압축 효율의 손실이 나타나기 시작하는 것을 볼 수 있다. 결과적으로 제안된 MV 예측 방법은 탐색 범위를 8픽셀 까지 감소 시키더라도 압축 효율의 저하는 나타나지 않으며, 67%의 IME 계산 복잡도 감소 성능을 얻을 수 있다. 클래스 B, C, D의 영상에 대해서는 계산 복잡도 감소 성능이 매우 높은 것을 볼 수 있는데, 이는 이 클래스의 영상들이 복잡한 움직임을 가지기 때문에 AMVP로는 정확하게 예측하기 어려운 MV들을 다수 포함하고 있기 때문이다. 다시 말하면 예측이 부정확한 MVP들로 인해 높은 IME 계산 복잡도가 필요로 되었지만, 상대적으로 정확한 제안된 MVP의 사용으로 인해 IME의 계산 복잡도가 큰 폭으로 감소할 수 있는 여지가 많았다는 것이다.

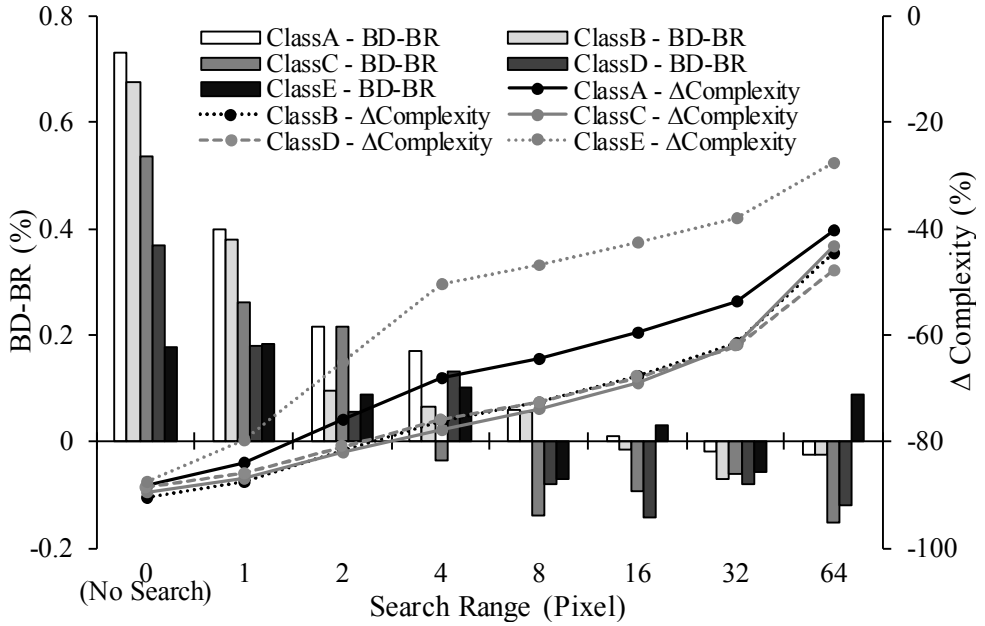


그림 3.7 제안된 MV 예측 방법을 적용한 단일 MVP기반의 IME의 계산 복잡도 변화와 압축 효율의 변화

표 3.4에는 3.3.1절에 제안된 MV 예측 방법을 적용한 복수 MVP 기반의 IME의 계산 복잡도 감소와 압축 효율의 변화를 나타내었다. 이 실험은 4장에서 후술 될 하드웨어 기반의 bottom-up IME에 제안된 bottom-up IME를 적용 할 경우를 고려하여 수행되었다. 즉, CTU 단위의 파이프라인을 고려하여, 이웃 PU의 움직임 정보가 결정되지 않아, AMVP를 정확하게 유도하지 못하는 상황에서의 bottom-up IME의 성능을 나타낸다. 이 때 사용 불가능한 이웃 PU의 움직임 정보를 대체하여 pseudo AMVP를 유도하여 사용하게 되는데 이 방법에 대해서는 6.2.2절에 나타나 있다. 표에는 4개의 좁은 탐색 범위에 대해 실험 한 결과를

나타내었다. 탐색 범위를 0으로 한 경우 계산 복잡도는 91.52%감소하며 1.01%의 BD-BR 증가를 보인다. 탐색 범위를 1 이상으로 증가시킴에 따라 압축 효율의 개선 폭은 점점 작아져 탐색 범위 4에서 포화된다. 반면에, 계산 복잡도는 탐색 범위가 늘어남에 따라 함께 지속적으로 늘어난다. 그러므로 탐색 범위는 압축 효율과 계산 복잡도 사이의 트레이드-오프 관계에 따라 0~4 사이의 값에서 적절히 선택 되어야 한다.

표 3.4 제안된 MV 예측 방법을 적용한 복수 MVP 기반의 IME의 계산 복잡도 감소와 압축 효율의 변화

Search Range		0 (No Search)		1		2		4	
Video Sequence		BR(%)	ΔC (%)	BR(%)	ΔC (%)	BR(%)	ΔC (%)	BR(%)	ΔC (%)
Class A	Traffic	0.89	-87.63	0.12	-72.11	0.05	-51.78	0.03	-40.23
	PeopleOnStreet	1.92	-91.62	1.08	-85.39	0.79	-76.98	0.63	-71.77
Class B	BasketballDrive	1.85	-93.30	0.58	-88.05	0.49	-80.77	0.25	-76.52
	BQTerrace	1.13	-90.29	0.16	-78.20	0.15	-67.96	0.04	-56.33
	Cactus	0.62	-90.62	0.23	-83.09	0.06	-72.34	0.10	-65.11
	Kimono	1.68	-92.24	0.17	-86.52	0.07	-78.73	0.00	-74.79
	ParkScene	0.64	-90.28	0.20	-79.05	0.13	-64.97	0.10	-58.37
Class C	BQMall	0.97	-90.76	0.41	-83.61	0.24	-73.83	0.14	-67.39
	BasketballDrill	1.07	-90.74	0.59	-84.19	0.19	-74.73	0.14	-67.95
	PartyScene	0.71	-88.36	0.40	-78.40	0.33	-65.96	0.34	-59.19
	RaceHorses	1.68	-92.25	0.77	-87.36	0.55	-80.78	0.43	-77.03
Class D	BasketballPass	0.71	-91.09	0.31	-83.87	0.09	-74.17	0.36	-68.81
	BlowingBubbles	0.53	-87.21	0.27	-75.57	-0.14	-61.47	-0.08	-54.22
	RaceHorses	1.38	-90.67	0.57	-84.72	0.39	-77.22	0.35	-73.10
Class E	FourPeople	0.36	-87.48	0.14	-73.82	0.08	-52.37	0.00	-35.40
	Johnny	0.69	-88.65	0.04	-74.03	0.34	-52.82	0.25	-37.67
	KristenAndSara	0.40	-89.85	0.10	-76.57	-0.28	-57.65	-0.15	-45.44
	Average	1.01	-90.18	0.36	-80.86	0.21	-68.15	0.17	-60.55

제 4 장 하드웨어 기반 Integer Motion Estimation

4.1 Bottom-up Integer Motion Estimation의 하드웨어 적용

4장에서는 앞서 3장에서 소개한 bottom-up IME를 적용한 하드웨어 기반의 IME를 설명한다. 제안된 하드웨어는 최하위 depth인 8x8 CU에 대해서는 TZS 알고리즘을 사용하여 IME를 수행하고 나머지 depth인 16x16 CU, 32x32 CU, 64x64 CU에 대해서는 탐색 범위가 1인 raster search 방법을 적용하도록 하였다. 이는 뛰어난 성능의 계산 복잡도 감소 효과와 함께 상대적으로 매우 적은 압축 효율을 가지는 지점을 고려하여 선택된 것이다. 비록 그림 3.6에서 탐색 범위를 1로 고정한 경우 평균 0.24%의 압축 효율 저하를 보이지만, 제안된 bottom-up IME 을 4장에서 하드웨어에 맞게 수정하는 과정에서 압축 효율 개선 효과를 얻을 수 있기 때문에 결과적으로 압축 효율의 손실은 발생하지 않는다.

최하위 depth가 아닌 CU에 대해서는 탐색 범위 1인 raster search 방법이 적용되기 때문에 하드웨어 구현이 매우 용이하다. 반면에 최하위 depth인 8x8 CU에 대해서는 소프트웨어 기반의 고속 IME 알고리즘인 TZS를 적용하기 때문에 앞서 2.3.1에서 설명한 하드웨어 기반의 IME에 고속 IME 알고리즘을 적용할 때 발생하는 하드웨어 사용률 저하 문제와 참조 데이터 접근 문제가 발생한다. 4장에서는 이러한 하드웨어 구현상의

이슈들에 대해 다루고, 이 이슈들을 해결하기 위한 방법을 함께 제시함으로써 bottom-up IME 알고리즘을 적용한 실시간 하드웨어 기반의 IME 구조를 제안한다.

4.2 하드웨어를 위한 수정된 Test Zone Search

4.2절에서는 8x8 CU에 TZS를 적용할 때 하드웨어의 특성을 고려하여 기존 TZS 알고리즘을 수정하는 방법을 설명한다. 본 절에서 수정된 TZS 알고리즘은 기존 TZS 알고리즘과 동일한 압축 효율 성능을 갖도록 설계되었다. 이러한 접근방식은 기존의 하드웨어 기반 IME에 대한 연구들이 기존의 고속 IME알고리즘의 압축 효율을 저하시키는 방식의 알고리즘 수정 방식과는 차별화된다.

4.2.1 SAD-tree를 활용한 CU 내 PU의 병렬 처리

SAD-tree는 앞서 2.3.1절에서 설명한 것과 같이 하드웨어 기반의 IME에서 다수의 블록 크기에 대한 IME를 병렬 처리하기에 적합한 구조이다. 본 논문에서 제안하는 하드웨어 기반의 IME에도 SAD-tree 구조를 참조하였다. SAD-tree는 SAD-tree 크기의 블록의 SAD 연산 뿐만 아니라, SAD-tree 크기의 블록 내에 포함되는 작은 블록 단위의 SAD 연산을 동시에 수행할 수 있다. 그러므로 8x8 CU를 위해 8x8 SAD-tree를 적용하면 8x8 CU 내에 존재하는 한 개의 2Nx2N PU, 두개의 2NxN PU, 두개의 Nx2N PU, 총 다섯 개의 PU에 대한 SAD 연산을 동시에 수행할 수 있다. 그러나, 소프트웨어 기반의 TZS 알고리즘은 각 PU 별로

독립적으로 TZS를 적용하기 때문에, 각 PU의 TZS 알고리즘의 수행 과정은 대부분의 경우 상이하다. 예를 들면, 다섯 개의 PU가 가지는 탐색 중심이 유도되는 AMVP에 따라 서로 다를 수 있으며, 이후 이어지는 diamond search 이후의 raster search의 수행 여부 또한 다를 수 있다. 게다가 refinement 단계의 반복 횟수 및 탐색 중심의 이동 위치 또한 PU 별로 상이할 것이다. 그러므로 TZS를 수행하기 위해서는 PU 단위로 고유한 TZS 알고리즘의 수행 스레드를 유지해야 하기 때문에, SAD-tree를 사용한 하드웨어 기반의 IME와 함께 사용되려면 TZS 알고리즘의 수정이 불가피하다.

본 논문에서는 TZS 알고리즘의 특성을 보존하면서도 SAD-tree 기반 IME의 이점을 활용하기 위해서 수정된 TZS 알고리즘을 제안한다. 그림 4.1에 8x8 CU를 위한 기존 TZS 알고리즘의 순서도와 함께 SAD-tree를 활용한 하드웨어를 위해 수정된 TZS 알고리즘의 순서도를 나타내었다. 그림 4.1 (a)는 기존 TZS알고리즘의 순서도를 나타낸다. 순서도의 가장 바깥쪽 루프는 TZS 알고리즘이 8x8 CU 내의 다섯 개 PU를 위해 반복되는 것을 나타낸다. 매 PU별로 AMVP를 유도한 뒤 2.2.1절에서 설명한 초기 diamond search, raster search, refinement search를 조건에 따라 순차적으로 수행하는 것을 볼 수 있다.

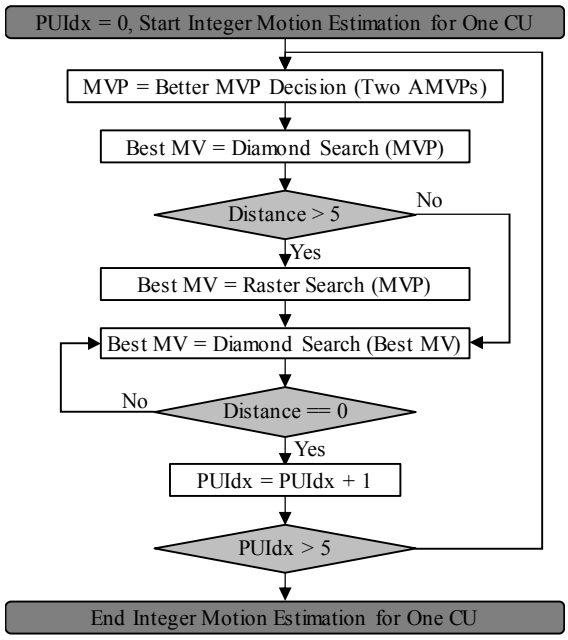
그림 4.1 (b)는 하드웨어를 위해 수정된 TZS 알고리즘을 나타낸다. 가장 큰 특징은 기존 TZS 알고리즘과 비교했을 때 8x8 CU 내의 다섯 개의 PU에 대한 루프를 제거한 점이다. 루프를 제거한 대신, 다섯 개의 PU는 모두 한번에 TZS 알고리즘의 각 단계를 거치게 된다. 가장 먼저 다섯 개의 PU로부터 각각 2개 씩의 AMVP를 유도하여 총 10개의 MVP 리스트를 구성하고 리스트 내의 동일한 MVP를 제거한다. 단, 이 때 두 개의 2NxN PU와 두 개의 Nx2N PU의 IME가 동시에 이루어지므로, 두

번째 2NxN PU의 상단 위치 (B0)와 두 번째 Nx2N PU의 좌측 위치 (A1)로부터 AMVP를 유도할 수 없게 된다. 그러므로 사용 가능하지 않은 위치의 AMVP는 각각 첫 번째 PU의 동일한 위치의 PU의 움직임 정보를 활용하도록 하였다. 예를 들면 두 번째 2NxN PU의 B0를 위해서는 첫 번째 2NxM PU의 B0 위치의 PU를 활용하는 것이다. 이러한 수정은 MV 예측의 정확도를 떨어트리는 원인이 될 수 있지만, 이로 인한 압축 효율의 저하는 평균 0.2% 안팎으로 매우 작다.

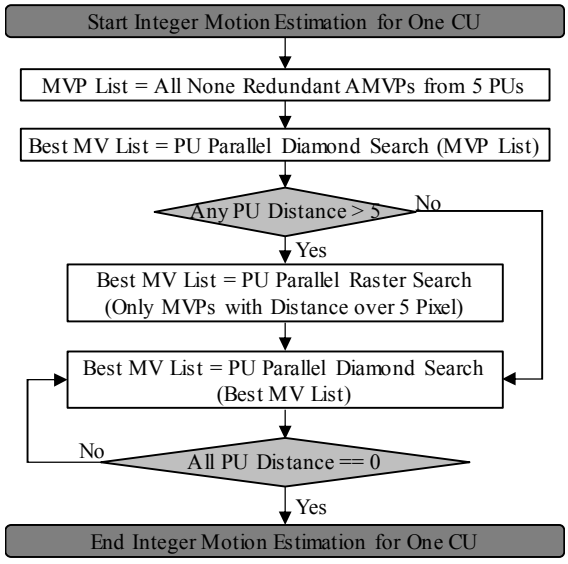
앞서 MVP 리스트가 구성되었으므로 이를 탐색 중심으로 하여 움직임 탐색을 수행하게 될 것이다. 단, 움직임 탐색을 수행할 때 수정된 TZS의 기존 TZS와 차별점이 명확하게 드러난다. 각 PU가 독립적으로 TZS에 의해 탐색 포인트를 결정하고 탐색을 수행하지만, SAD-tree가 사용되므로 탐색 되는 포인트들에서 모든 PU의 SAD 값이 계산되는 점이다. 그러므로 서로 다른 PU는 독립적인 TZS 알고리즘의 스레드를 가진 채로 동작하지만, PU들은 서로의 탐색 포인트를 공유하며, 서로 다른 PU의 탐색 포인트로부터도 자신의 최적 MV를 찾을 수 있다. 그러므로 다른 PU의 탐색 포인트로부터 새로운 탐색 중심을 찾아낼 수도 있다. 이러한 차별점은 탐색 과정 전반에 모두 적용된다.

다음으로 유도된 MVP 리스트 중에서 하나의 MVP를 선택하여 탐색 중심을 결정하지 않고 모든 MVP를 사용하여 초기 diamond search를 수행한다. 다음으로 모든 MVP에 대한 초기 diamond search가 종료되면, raster search 수행 여부를 판단하기 위해 각 PU의 탐색 중심과 초기 diamond search 를 통해 찾아낸 새로운 최적 MV 간의 거리를 PU 별로 독립적으로 구한 다음 하나의 PU라도 raster search의 수행 조건을 만족하면 raster search 단계로 진행한다. 모든 PU가 raster search의 수행 조건을 만족하지 못한 경우에는 refinement search로 바로 진행한다. Raster

search 과정에서는 초기 탐색 중심과 새로 찾아낸 최적 MV간의 거리가 5를 초과하는 PU의 최적 AMVP를 중심으로 하여 64 픽셀 의 탐색 범위로 raster search를 수행한다. 이 때에도 역시 모든 탐색 포인트들에 대해 모든 PU의 SAD값이 계산되어 새로운 최적 MV를 업데이트 하게 된다. 모든 PU에 대해서 raster search가 종료되거나, 초기 diamond search 에서 모든 PU가 raster search를 수행할 필요가 없는 조건이 만족되면, refinement search 단계로 진행한다. Refinement search 단계에서는 앞 단계에서 업데이트된 새로운 최적 MV를 탐색 중심으로 하여 diamond search를 수행하는 과정을 반복한다. 이 반복 과정은 다섯 개의 PU가 갖는 고유의 탐색 중심에 대해 한 번씩 모두 diamond search를 수행한 다음 모든 PU의 최적 MV를 업데이트 하고, 업데이트된 각 PU의 최적 MV를 새로운 탐색 중심으로 하여 diamond search를 수행하는 방식으로 이루어진다. 이 반복 과정에서도 또한 다섯 개 PU의 TZS 스레드가 독립적으로 생성하는 탐색 포인트들이 모든 PU들에게 공유된다. 각 PU의 TZS 스레드는 더 이상 최적 MV가 발견되지 않으면 중단되어 더 이상 새로운 탐색 포인트를 생성하지 않을 수 있지만, 다른 PU의 TZS 스레드의 탐색 포인트에서 새로운 최적 MV가 발견되면, 중단되었던 PU의 TZS 스레드는 새로운 최적 MV를 탐색 중심으로 하여 다시 refinement search를 수행하게 된다. 그러므로 수정된 TZS의 refinement search가 종료되기 위해서는 한 번의 반복 과정에서 생성된 탐색 포인트에 대해서 모든 PU들에 대한 새로운 최적 MV가 더 이상 발견되지 않아야 한다. 이 refinement search의 종료 조건을 만족하고 나면, 하나의 CU에 대한 수정된 TZS 알고리즘을 사용한 IME가 종료된다.



(a)



(b)

그림 4.1 (a) 8x8 CU를 위한 기존 TZS 알고리즘의 순서도, (b) SAD-tree를 활용한 하드웨어를 위해 수정된 TZS 알고리즘의 순서도

스케줄에 대한 이해를 돕기 위해 표 4.1에 diamond search의 반복 단계 별 다섯 개의 PU 스레드 동작의 예를 나타내었다. 첫 반복 단계에서는 다섯 개의 PU가 고유의 스레드를 가지고 탐색을 수행한다. 단, 서로의 탐색 중심은 일치할 수도 일치하지 않을 수도 있다. 첫 번째 반복 탐색 결과 2NxN_1 PU에 대한 새로운 최적 MV가 발견되지 않아서 2NxN_1 PU를 위한 TZS 스레드는 종료된다. 마찬가지로 두 번째 반복 탐색에서는 2Nx2N, Nx2N_1 PU의 스레드가 종료된다. 그러나 세 번째 반복 탐색 결과 2NxN_0 및 Nx2N_0 PU의 스레드에 의해 탐색된 탐색 지점에서 2NxN과 Nx2N_1 PU의 새로운 최적 MV가 발견되어 이 PU들의 스레드는 다시 시작된다. 반면 2NxN_0와 Nx2N_0 스레드는 새로운 최적 MV가 발견되지 않아 중단된다. 네 번째 반복 탐색 결과 또다시 2Nx2N과 Nx2N_1의 새로운 최적 MV가 발견되지 않아 중단되고, 새로 2NxN_1 PU의 새로운 최적 MV가 발견되어 스레드가 다시 시작된다. 다섯 번 째 반복 탐색에서 앞서 새로 시작된 2NxN_1 PU의 스레드에 의해 탐색이 수행되지만 어떤 PU에 대해서도 새로운 최적 MV를 찾지 못한다. 이 경우 여섯 번째 반복 탐색 과정은 시작되지 않고 TZS는 종료된다.

표 4.1 수정된 TZS의 Diamond search의 반복 수행의 예

Diamond search iteration index	Search center for each PU					Note
	2Nx2N	2NxN_0	2NxN_1	Nx2N_0	Nx2N_1	
0	(3,3)	(3,3)	(8,7)	(-22,-12)	(-22,-12)	2NxN_1 finished
1	(4,5)	(4,7)	none	(-24,-14)	(-24,-14)	2NxN, Nx2N_1 finished
2	none	(7,7)	none	(-24,-18)	none	2NxN_0, Nx_2N_0 finished, 2NxN, Nx2N_1 restart
3	(9,12)	none	none	none	(-24,-24)	2Nx2N, Nx2N_1 finished, 2NxN_1 restart
4	none	none	(12,12)	none	none	2NxN_1 finished
5	none	none	none	none	none	No more new search center → Diamond search finished

4.2.2 Grid 기반의 Sampled Raster Search

기존 TZS는 각 PU마다 서로 다른 raster search 탐색 영역을 설정하여 raster search를 독립적으로 수행한다. 그러나, 수정된 TZS의 경우 다섯 개의 PU가 에 대한 raster search가 동시에 수행되므로 본 논문에서는 수정된 TZS를 위한 raster search의 탐색 범위를 새롭게 정의하였다. 그림 4.2에 수정된 TZS의 raster search를 위한 탐색 범위 설정 방법의 예를 나타내었다. 우선 사용 가능한 전체 탐색 영역을 일정한 크기의 격자로 나누었다. 이 예의 경우 가로 256 픽셀, 세로 176 픽셀이 다섯 개의 PU가 사용 가능한 탐색 영역임을 나타낸다. 각 격자는 raster search를 수행하는 단위를 나타내며, 이 예의 경우 각 격자의 크기는 가로 및 세로 방향으로 모두 16픽셀이다. 격자 위에 표시된 다섯 개의 반투명한 흰 색 사각형은 각 PU의 raster search의 탐색 범위를 나타낸다. 각 PU의 raster search 탐색 범위는 ±64픽셀이므로 이 때 수정된 TZS를 위한

raster search 탐색 범위는 다섯 개 PU의 탐색 범위가 포함되어 있는 모든 격자로 설정된다. 이 탐색 범위를 빗살무늬 격자로 나타내었다. 이렇게 설정된 탐색 범위에서 수정된 TZS는 raster search를 수행하게 된다. 수정된 TZS 또한 기존의 TZS와 같이 가로 세로 방향으로 샘플링 된 탐색 지점에 대해서 탐색하는데, 이 샘플링은 매 격자 단위로 동일하게 이루어져 있다. 격자 기반의 탐색 범위 설정은 다음 절에서 서술될 중복 연산 제거를 위해 사용되며, 구체적 탐색 방법은 4.5절의 하드웨어 구조 부분에 자세히 나타나 있다.

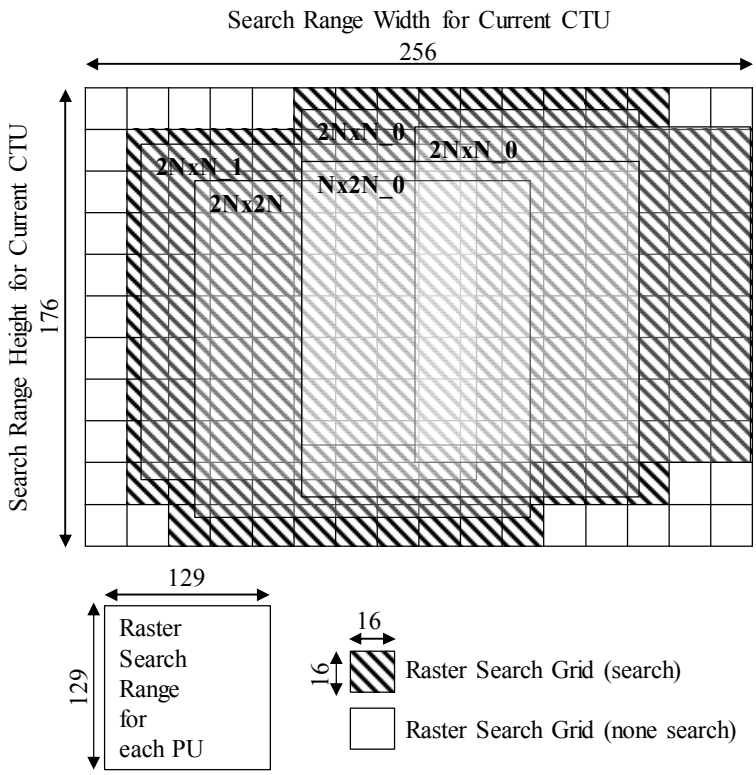


그림 4.2 수정된 TZS의 raster search를 위한 Grid 기반의 탐색 범위 설정 방법

4.2.3 서로 다른 PU 간의 중복 연산 제거

수정된 TZS는 초기 diamond search, raster search 및 refinement search의 크게 세 가지 단계로 구성되어 있다. 또한 refinement search는 반복 수행이 가능하여 세 가지 이상의 단계로 수정된 TZS는 수행될 수 있다. 그런데, 수정된 TZS는 각 PU의 TZS 알고리즘 스레드를 고유하게 가지고 탐색을 수행하기 때문에, 각 PU의 TZS 알고리즘이 같은 단계에서 동일한 탐색 지점을 생성하는 경우가 발생할 수 있다. 예를 들면 raster search 단계에서 다섯 개의 PU가 서로 다른 AMVP로 탐색 범위를 설정하더라도, 넓은 범위를 raster search의 탐색 범위로 설정하기 때문에 매우 높은 확률로 각 PU의 넓은 범위의 탐색 영역이 겹치게 된다. 이러한 탐색 영역을 각 PU 별로 독립적으로 모두 수행하는 경우, 높은 비율의 중복 연산을 야기할 것이다. 또한 refinement search의 경우에도 각 PU가 각자의 TZS 알고리즘 스레드에 따라 탐색 중심을 설정하고 diamond search를 수행하더라도, 경우에 따라 서로 다른 PU의 탐색 중심이 일치하는 경우가 나타날 수 있다. 이는 CU 내 PU들의 움직임이 공통된 경우 매우 빈번하게 일어난다. 이 경우 또한 각 PU의 탐색 중심에 대해 diamond search를 모두 수행하면 동일한 탐색 중심을 갖는 PU들 간의 연산은 중복 연산을 초래한다. 그러므로 이 절에서는 이러한 중복 연산을 제거하기 위한 방법을 제안한다.

Raster search의 중복 연산의 경우, 각 PU의 raster search 영역을 모두 파악한 뒤 이 영역들에 대해서 단 한번만의 탐색을 수행하도록 하면 중복 연산을 막을 수 있다. 각 PU가 발생 시키는 raster search의 탐색 영역은 매우 넓기 때문에 이 영역에서 중복되는 영역을 구분하는 데에는 비교적 많은 연산이 필요하다. 그러나, 이는 앞서 4.2.2절에서 설명한

격자 기반의 raster search 탐색 방법을 사용함으로써 쉽게 달성될 수 있다. 다음으로 diamond search의 중복 연산의 경우, 각 PU의 탐색 중심을 모두 파악 한 뒤, 마찬가지로 각 탐색 중심에서 단 한번만의 diamond search를 수행하도록 하면 중복 연산을 막을 수 있다. 넓은 영역을 고려해야 하는 raster search 와 다르게 최대 다섯 개의 탐색 중심에 대해서만 비교 연산을 수행하여 중복된 탐색 중심을 찾아낼 수 있으므로, 비교적 간단히 중복 연산을 제거할 수 있다. 탐색 중심이 아닌 탐색 지점 단위로 중복 연산을 제거하는 방법을 고려할 수 있지만, 4.4.4절에 후술 될 과도한 하드웨어 리소스 사용의 문제로 인해, 탐색 중심의 중복을 제거하는 방법을 선택하였다.

4.3 Idle cycle이 감소된 5-stage 파이프라인 스케줄

4.3.1 파이프라인 스테이지 별 동작

이 절에서는 고속 동작이 가능한 하드웨어 기반의 IME를 위해 5-stage의 파이프라인 동작을 제안하고 각 스테이지 별 동작에 대해 설명한다. 본 논문에서 제안된 하드웨어는 각 CU크기별 시분할 다중화 방식을 사용하고 있다. 제안된 하드웨어는 총 8개의 16x16 SAD tree를 사용한다. 그러므로 한 번에 32개의 8x8 블록 크기의 탐색 지점을 탐색 할 수 있으며, 8개의 16x16 블록 크기의 탐색 지점을 탐색 할 수 있다. 32x32 및 64x64 블록 크기의 경우에는 16x16 SAD를 4 차례 및 16차례 축적하여 8개의 탐색 지점에 대한 탐색을 수행한다. 표 4.2에 다섯 개의 파이프라인 스테이지의 정의와 해당 스테이지에서 수행되는 동작에 대해

정리하여 나타내었다. 먼저 첫 번째 파이프라인 스테이지인 'A' 스테이지에서는 IME 하드웨어 전체를 스케줄링하고, 현재 IME를 수행할 CU, PU, 참조 픽쳐 및 탐색 지점을 결정한다. 결정된 탐색 지점에 대해 블록 매칭 연산을 수행하기 위해 탐색 지점에 해당하는 참조 데이터가 저장된 참조 메모리의 주소를 계산한다. 'F' 스테이지는 'A' 스테이지에서 계산한 참조 메모리 주소에 접근하여 참조 데이터를 내부 메모리로부터 fetch하는 스테이지이다. 다음으로 'C' 스테이지는 앞서 fetch된 참조 데이터와 원본 데이터간의 SAD 값을 계산하고, 탐색 지점을 가리키는 MV의 bit cost를 계산하는 스테이지이다. 단, 'C' 스테이지에 SAD 연산이 집중되어 동작 주파수가 낮아지는 현상을 방지하기 위해 SAD값은 16x16 단위 까지만 계산된다. 마찬가지로 다음 'b' 스테이지에 비교 연산이 집중되는 것을 방지하기 위해 계산이 완료된 8x8 CU의 32개의 탐색 지점 중 더 나은 16개를 선택하는 비교 연산이 이 스테이지에서 먼저 이루어진다. 다음으로 'b' 스테이지에서는 'C' 스테이지에서 16x16 SAD까지 계산된 값을 축적하여 32x32 및 64x64 SAD를 생성하며, 모든 CU 크기에 대한 최적 MV를 결정한다. 마지막으로 'B' 스테이지에서는 최적 참조 픽쳐, 최적 PU 크기 및 최적 CU depth 결정을 수행한다. 제안된 하드웨어 기반 IME는 CTU 단위의 파이프라인 스테이지 중 하나임을 가정하였기 때문에, SAD 기반의 RD cost가 모든 최적 모드 결정에 사용된다. 최적 CU depth 결정까지 이루어져야 하는 이유는, AMVP를 유도하기 위한 것으로, 이 AMVP는 실제 최종 AMVP와는 다를 수 있지만, 탐색 중심을 결정하기 위한 것으로 다소의 오차는 HEVC 표준 준수에 문제가 되지 않는다.

표 4.2 제안한 하드웨어 기반 IME의 5-stage 파이프라인 스테이지 및 각 스테이지 별 동작

Name	Abbreviation	Operation
Address	A	Reference memory address generation, Scheduling (CU, PU, reference picture, search point)
Fetch	F	Reference memory access
Cost	C	8x8, 16x16 SAD Calculation, Bit cost Calculation, 16 of better MV decision for 8x8 CU
Best1	b	32x32, 64x64 SAD Calculation, Best MV decision for all CU size
Best2	B	Best reference picture decision, Best PU type decision, Best CU depth decision

4.3.2 Test Zone Search의 의존성으로 인한 Idle cycle 도입

앞서 2.3.1절에서 설명한 바와 같이 고속 IME 알고리즘은 여러 단계로 구성이 되어 있으며, 각 단계의 탐색 결과가 다음 단계의 탐색에 영향을 미치는 의존성을 지니고 있다. 때문에 이러한 의존성은 하드웨어 기반 IME의 파이프라인 스케줄에 idle cycle을 발생시켜 하드웨어의 사용률을 떨어트리고, 결과적으로 고속의 IME를 수행할 수 없게 되는 문제를 야기한다. 이는 고속 IME알고리즘은 TZS의 경우에도 마찬가지로 발생하는 문제로써, 하드웨어의 사용률을 크게 저하시키게 된다. 그림 4.3에 본 논문에서 채택한 5-stage IME 파이프라인 스케줄을 사용하였을 때 TZS의 전 단계 의존성으로 인한 idle cycle 발생의 예를 나타내었다. 가로 방향은 시간의 흐름을 나타내며, 정사각형 내의 영문 알파벳은 4.3.1절에서 나타낸 다섯 개의 파이프라인 스테이지를 나타낸다. 이 예에서는 raster search는 사용되지 않았으며, 각 단계별로 두 개의 고유한

탐색 중심으로 탐색을 수행하고 있다. 그러므로 각 단계에서 2 cycle에 걸쳐 두개의 탐색 중심을 탐색한다. 또한 첫 번째 8x8 CU는 세 단계의 탐색으로 TZS가 종료되고, 두 번째 8x8 CU는 두 단계의 반복 탐색으로 TZS가 종료되는 것을 가정하였다. 그림에서 A로부터 시작된 파이프라인 스케줄이 B 스테이지까지 도달하고 난 다음에서야 최적 MV가 완전히 결정되기 때문에 다음 반복 단계의 탐색 중심을 알 수 있는 시점은 B 스테이지가 되므로 다음 반복 단계의 A스테이지는 전 반복 단계의 B 스테이지가 종료되고 난 다음에야 가능한 것을 볼 수 있다. 이 예에서는 총 다섯 개의 idle cycle 그룹이 나타나는데 각 그룹 별로 4 cycle의 idle cycle이 나타난다. 이는 5-stage 파이프라인 스케줄을 사용하였기 때문이다. 그림에서 나타낸 것과 같이 고속 IME 알고리즘을 사용하는 경우 단계별 의존성으로 인해 하드웨어의 실질적 사용률이 약 33%에 그치게 되고 이는 하드웨어의 리소스 낭비와 throughput 저하로 이어지기 때문에 고성능의 하드웨어 기반 IME를 설계하는 데에 큰 장애물이 된다.

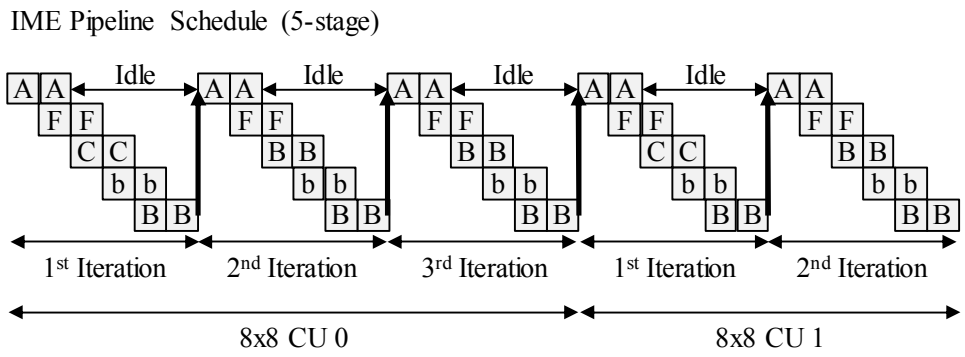


그림 4.3 5-stage IME 파이프라인 스케줄을 사용하였을 때 TZS의 전 단계 의존성으로 인한 idle cycle 발생의 예

4.3.3 컨텍스트 스위칭을 통한 Idle cycle 감소

앞서 살펴본 TZS로 인한 하드웨어 IME의 사용률 저하 문제는 이전 연구들이 제안했던 방법과 같이 각 반복 단계 간의 의존성을 제거함으로써 해결될 수 있지만, 이 경우 고속 IME 알고리즘의 성능을 온전히 보존할 수 없게 된다. 그러므로, 본 논문에는 고속 IME 알고리즘의 성능을 훼손하지 않으면서도 idle cycle이 없는 고속 IME 하드웨어를 실현하기 위해서 서로 의존 관계가 없는 탐색 스레드 간의 컨텍스트 스위칭을 통한 idle cycle의 제거 방법을 제안한다.

4.2.1에서 제안된 수정된 TZS에서 하나의 PU 스레드가 생성하는 탐색 지점은 하나의 PU 뿐만 아니라 서로 다른 PU의 다음 반복 단계의 탐색 중심 결정에도 영향을 미치기 때문에 다섯 개의 PU가 모두 서로 밀접한 의존관계를 가지고 있다. 그러나, 이는 한 장의 참조 픽쳐 사용이 가정된 경우이며, 공통 테스트 조건과 같이 4 장의 참조 픽쳐가 사용되는 경우에는 참조 픽쳐 간에는 어떠한 의존성도 존재하지 않는다. 서로 다른 depth의 PU들 또한 서로의 탐색 지점 결정에 간섭하지 않기 때문에 독립적이라고 할 수 있다. 그러므로 본 논문에서는 이러한 독립적인 탐색 스레드들을 컨텍스트 스위칭 방법을 통해 스케줄 하도록 하였다.

표 4.2에 하드웨어 기반의 IME를 위한 컨텍스트의 종류 및 내용을 나타내었다. 최하위 depth인 8x8 CU와 그렇지 않은 나머지 CU크기에 대해 서로 다른 컨텍스트가 사용된다. 컨텍스트의 종류는 2가지 이지만, 각 depth 0, 1, 2 별로 하나의 컨텍스트가 사용되며, 8x8 CU의 경우 각 참조 픽쳐 별로 컨텍스트가 사용된다. 즉, 총 7개의 IME 스레드가 번갈아가면서 IME를 수행하게 되는 것이다. 각 컨텍스트의 내용은

사용된 IME 알고리즘에 따라 달라진다. 8x8 CU가 아닌 나머지 CU들에 대해서는 BMVP 탐색 범위가 1인 정적인 raster search 가 사용되므로 비교적 컨텍스트의 내용이 간단하다. 현재 IME를 수행중인 참조 픽쳐의 인덱스, 사용 가능한 BMVP가 존재하는지에 대한 여부, 현재 탐색을 수행중인 BMVP의 인덱스와 현재 SAD를 구하려는 CU내 16x16 영역의 인덱스를 나타낸다. 반면 8x8 CU를 위한 컨텍스트는 TZS 스레드의 현재 상태를 저장해야 하기 때문에 조금 더 복잡한 정보를 가지게 된다. 현재 TZS가 어떤 단계에 포함되어 있는지, 현재 raster search가 수행되는 격자의 인덱스, 다음 탐색 중심 결정을 위한 현재 최적 MV의 cost및 MV, 현재 스레드 자신이 idle 상태인지 아닌지에 대한 플래그가 포함되어 있다. 이러한 컨텍스트 정보를 스케줄러에서 관리하면서, 현재 idle 상태인 스레드를 대신하여, 현재 수행 가능한 스레드를 선택하여 IME를 수행하도록 함으로써 idle cycle을 제거할 수 있다.

표 4.3 제안된 컨텍스트 스위칭을 위한 컨텍스트 목록 및 내용

Context Type	Context	Contents
ctxn8	Depth0, Depth1, Depth2	Reference index / BMVP valid / BMVP index / SAD partition index
ctx8	Depth3_Ref0, Depth3_Ref1, Depth3_Ref2, Depth3_Ref3	TZS phase (1st diamond search or raster search or diamond search) / Current raster search grid index / PU thread valid, current best cost and current best MVs, for 5 PUs

제안된 컨텍스트 스위칭 방법을 통해 IME를 수행한 경우의 스케줄 예시를 그림 4.4에 나타내었다. 모든 조건은 그림 4.3에 서술한 바와 동일하며, 그림 4.3의 예시에 컨텍스트 스위칭을 활용한 스케줄을 적용하였을 때의 스케줄 변화를 나타내었다. 기존 8x8 CU의 참조 픽쳐 0의 수행으로 인해 발생했던 idle cycle에 다른 컨텍스트의 IME 동작을 interleaving함으로써 idle cycle을 제거할 수 있음을 확인할 수 있다.

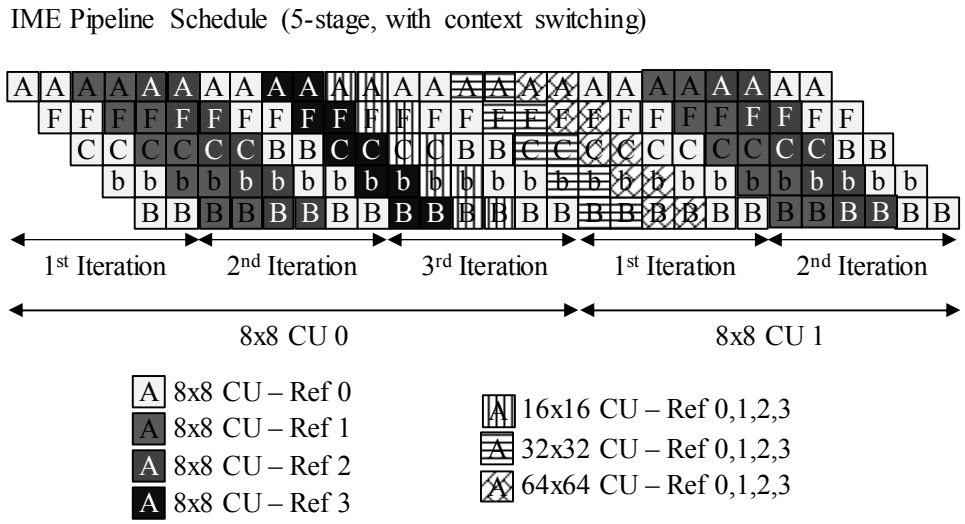


그림 4.4 제안된 컨텍스트 스위칭 방법을 적용한 IME 스케줄의 예

4.4 고속 동작을 위한 참조 데이터 공급 방법

4.4.1 참조 데이터 접근 패턴 및 접근 지연 발생 시 문제점

하드웨어 기반 IME의 고속 동작을 위해서는 4.3절에서 살펴본 바와 같이 알고리즘의 의존성을 해결하여 IME 하드웨어의 높은 사용률을 유지하는 것이 중요하다. 그러나, IME 하드웨어가 높은 사용률을 유지하기 위해서는 IME 연산에 필요한 입력 데이터를 원활하게 공급할 필요가 있다. IME 하드웨어가 모든 의존성을 극복하고 매 cycle 마다 연산을 수행할 수 있도록 설계되었다고 하더라도, 연산을 위한 참조 데이터가 제때 공급되지 못하면, 참조 데이터를 준비하는 동안 IME 하드웨어가 idle 상태가 되는 문제가 나타나게 된다. 2.3.1절에서 살펴본 것과 같이, 고속 IME 알고리즘은 이전 단계의 탐색 결과가 다음 단계의 탐색 전략에 영향을 미치기 때문에 다음 단계의 탐색을 위한 참조 데이터를 pre-fetch하는 것이 어렵다. 또한 diamond search 패턴이나, raster search 패턴 등 탐색 방법에 따라 상이한 참조 데이터를 요구하기 때문에 요구되는 참조 데이터를 원활하게 fetch하는 것이 어렵다.

그림 4.5에 참조 데이터의 fetch가 원활하게 이루어지지 못하는 경우 IME 하드웨어 스케줄의 예를 나타내었다. 이 예에는 4.3.3절에 소개된 컨텍스트 스위칭을 적용하지 않은 경우를 나타내었다. 파이프라인 스케줄에서 필요로 하는 참조데이터가 제때 공급되지 못하고, 여러 cycle의 지연이 발생하는 경우, 이어지는 파이프라인 스테이지인 ‘C’, ‘b’, ‘B’ 스테이지가 동작되지 못하여 하드웨어가 idle 상태가 되는 것을 볼 수 있다. 이러한 참조 데이터 fetch의 지연이 자주 발생할수록 하드웨어의 사용률은 저하되고, IME 하드웨어의 throughput 또한

감소한다.

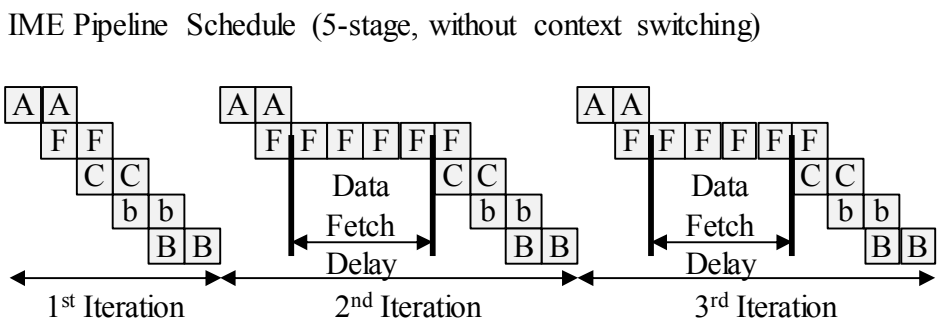


그림 4.5 참조 데이터 접근에 지연이 발생하는 경우 IME 하드웨어 스케줄의 예

4.4.2 Search Points의 Locality를 활용한 참조 데이터 접근

제안된 하드웨어 기반의 IME는 단일 cycle에 32개의 8x8 CU에 대한 탐색 지점을 탐색하고 있다. 그러므로 한 번에 32개의 서로 다른 지점의 참조 데이터를 fetch해야 4.4.1절에 나타낸 참조 데이터 접근 지연으로 인한 하드웨어 사용률 저하를 막을 수 있다. 그러나, 32개의 서로 다른 참조 데이터 영역을 fetch하기 위해서 32개의 port를 갖는 메모리를 사용하는 것은 불가능하다. 그러므로 이 절에서는 32개의 탐색 포인트에 대한 참조 데이터를 fetch 하기 위해 참조 데이터의 locality를 활용한다.

TZS의 diamond search 패턴은 탐색 중심근처에는 조밀하게 탐색을 수행하고 탐색 중심으로 멀어질수록 듬성듬성 탐색을 수행하도록 탐색 패턴을 구성하고 있다. 특히 탐색 중심으로부터 거리가 8픽셀을 을

초과한 탐색 지점에 대해서는 각 거리 별로 4개의 탐색 지점만을 탐색하도록 되어있다. 그림 4.6에 8픽셀까지 거리의 diamond search 패턴과 각 탐색 지점에서 8x8 CU를 위해 필요한 참조 데이터의 영역을 나타내었다. 대부분의 탐색 포인트가 탐색 중심에 몰려있고, 8 픽셀의 거리를 갖는 부분만 상대적으로 멀리 떨어져 있는 것을 볼 수 있다. 또한 서로의 필요 참조 데이터가 중복되어 있는 것을 볼 수 있다. 이러한 참조 데이터의 locality는 8픽셀 거리까지의 diamond search 패턴에서만 나타나며, 다음 탐색 지점 거리인 12, 16, 24, 32, 48, 64 픽셀 거리에 대해서는 서로의 탐색 지점의 거리가 멀어져 거의 locality를 보이지 않게 된다. 그러므로 locality를 갖는 8픽셀 거리까지의 diamond search 패턴이 필요로 하는 참조 데이터를 한 번에 fetch해 오는 전략을 사용한다. 즉, 그림 4.6에 나타낸 회색으로 음영 처리된 영역의 참조 데이터를 한 번에 fetch한 다음, 각 탐색 지점을 위한 참조 데이터를 각 SAD-tree에 배포하는 방식을 사용하였다. 마찬가지로 Raster search 의 참조 데이터 또한 인접한 32개의 탐색 포인트들이 필요로 하는 참조 데이터를 한 번만 fetch 하여 각각 탐색 지점의 SAD-tree에 배포하도록 하였다.

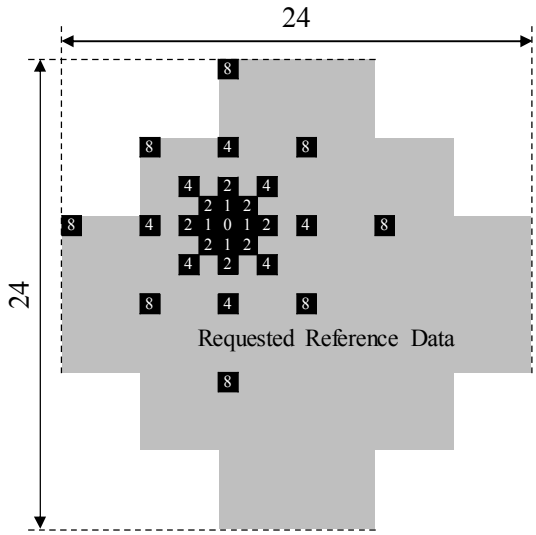


그림 4.6 8픽셀 거리까지의 diamond search 패턴과 이 때 필요한 참조 데이터 영역

4.4.3 단일 cycle 참조 데이터 접근을 위한 Multi Bank 메모리 구조

제안된 IME 하드웨어는 매 cycle 마다 하나의 탐색 중심에 대한 32개의 탐색 지점을 탐색해야 하므로, 단일 cycle에 넓은 범위의 참조 데이터를 fetch 할 수 있어야 한다. 그림 4.6에 나타낸 diamond search 패턴을 위한 참조 데이터뿐 만 아니라 raster search에 필요한 참조 데이터를 fetch 하려면 24x24 pixel 데이터를 단일 cycle에 fetch 할 수 있어야 한다. 이 때 대역폭은 576 byte/cycle로 일반적인 내부메모리로 사용되는 SRAM의 포트 크기인 수 byte보다 월등히 넓은 대역폭이다. 그러므로 제안된 IME 하드웨어를 위해 576 byte/cycle의 넓은 대역폭을 구현하기 위해서 multi bank 메모리 구조를 사용하였다. 그림 4.7에 multi

bank를 활용하여 넓은 범위의 참조 데이터를 단일 cycle에 fetch 하는 예를 나타내었다. 이 예에서는 총 여섯 개의 SRAM이 사용되었다. 참조 픽쳐를 고유의 음영 처리된 영역과 같이 배열하여 여섯 개의 SRAM에 나누어 저장하면, 그림에 굵은 테두리로 표시된 사각형들과 같이 넓은 영역의 참조 데이터가 요청되더라도, 단일 cycle에 SRAM 여섯 개로부터 참조 데이터를 fetch 할 수 있다.

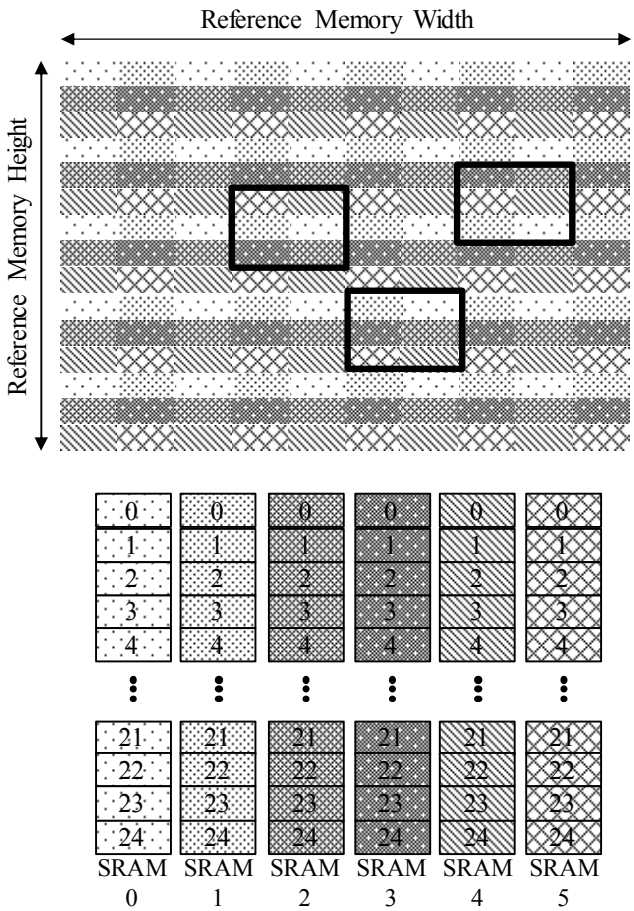


그림 4.7 Multi bank SRAM을 활용하여 넓은 영역의 참조 데이터를 단일 cycle에 fetch 할 수 있는 메모리 구조

이와 같이 multi bank 메모리 구조를 활용하여 넓은 영역의 참조 데이터를 fetch 하는 방법은 과거 연구에서도 사용된 방법이다 [34,36]. 제안된 하드웨어를 위해서는 64bit의 대역폭을 갖는 96개의 SRAM을 활용한 multi bank 메모리 구조가 사용되었다. 세부 구조와 활용방법에 대해서는 4.5절 의 하드웨어 구조에서 자세히 설명하였다.

4.4.4 참조 데이터 접근의 자유도 제어를 통한 스위칭 복잡도 저감 방법

4.4.3절에서 multi bank 메모리구조를 통해 24x24 픽셀 영역을 단일 cycle에 fetch 할 수 있도록 하였다. 다음으로 fetch된 데이터를 32개의 8x8 SAD-tree로 적절히 배분하여 공급해야 하지만, 이 때 SAD-tree와 fetch된 24x24 픽셀 영역 중 임의의 32개의 8x8 픽셀 영역을 선택하여 SAD-tree로 공급하는 과정에서 대량의 Mux 로직이 사용될 수 있다. 그림 4.8에 fetch된 24x24 픽셀 영역의 참조 데이터가 여러 탐색 지점에 의해 사용 될 때 이를 SAD-tree로 공급하려는 경우의 예를 나타내었다. 만약 몇 개의 탐색 중심이 서로 인접하여 탐색 지점이 그림과 같이 나타나는 경우 이 탐색 지점을 위해 필요한 참조 데이터를 SAD-tree로 공급하기 위한 Mux가 SAD-tree와 fetch된 데이터 사이에 필요하다. 이 Mux의 스위칭 복잡도는 식 (4.1)과 같이 나타난다. 이 Mux를 구성하기 위한 하드웨어 리소스를 추산해 보면 다음과 같다. 16x16 픽셀 영역으로부터 한 개의 탐색 지점을 선택하기 위해서는 256 to 1 Mux가 필요하며, 하나의 8x8 SAD-tree로 64개의 8bit 픽셀을 보내기 위해서는 1bit의 256 to 1 Mux가 512개 필요하게 된다. 또한 32개의 8x8 SAD-tree가 존재하므로, 1bit의 256 to 1 Mux가 16,384개가 필요하게 되며, 이 Mux를 gate count

단위의 하드웨어 리소스로 환산하면 약 5.6M gate가 된다. 이는 큰 크기의 참조 데이터 버퍼에서 지나치게 자유로운 참조 데이터 접근을 허용했기 때문에 발생하는 문제이다. 이는 기존 하드웨어 기반 IME에서는 작은 크기의 참조 데이터를 버퍼를 사용하였고, 다수의 탐색 지점을 자유롭게 접근하여 한 cycle에 처리하려는 시도를 하지 않았기 때문에 나타나지 않았던 문제이다.

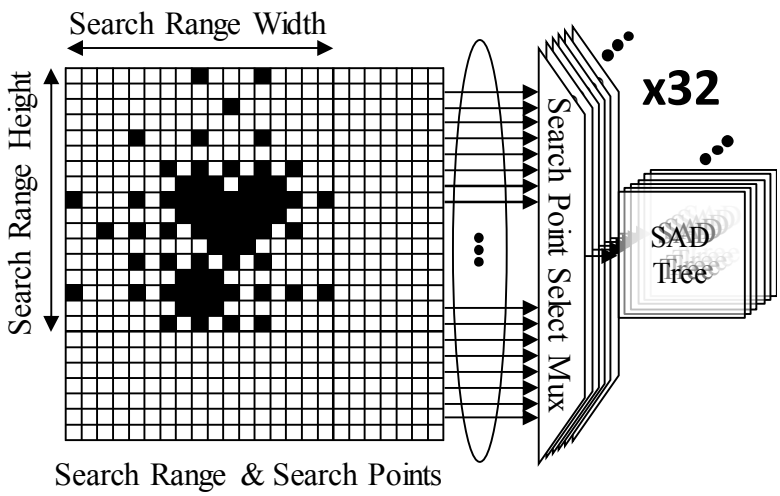
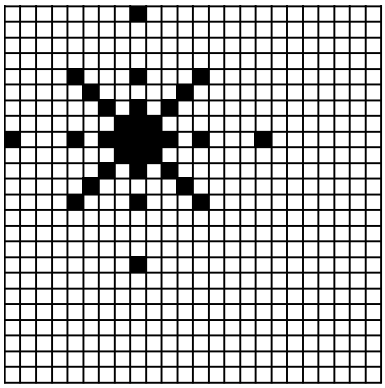


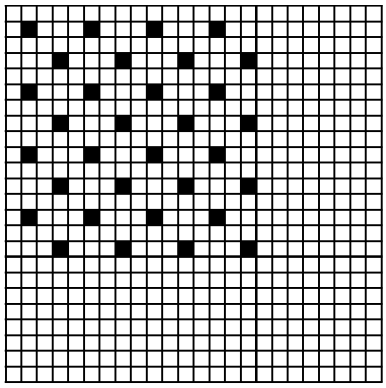
그림 4.8 참조 데이터 버퍼로부터 SAD-tree로 특정 탐색 지점의 참조데이터를 공급하기 위한 하드웨어 구조

$$\begin{aligned}&\text{Search Point Select Mux's Switching Complexity}\\&= (\text{Search Range Width} \times \text{Search Range Height})\\&\times (\text{SAD Tree Width} \times \text{SAD Tree Height})\\&\times (\text{Number Of SAD Tree})\end{aligned} \tag{4.1}$$

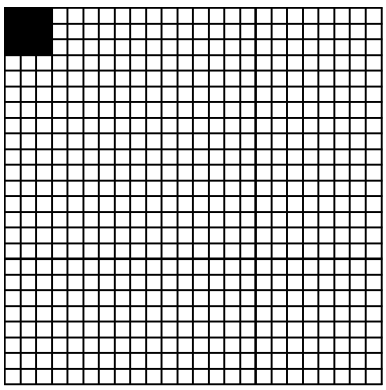
앞서 설명한 바와 같이 높은 자유도를 갖는 스위칭 로직은 많은 하드웨어 리소스를 필요로 하게 되므로, 참조 데이터 접근의 자유도를 제한하여 하드웨어 리소스의 사용량을 줄여야 한다. 그림 4.7에 나타낸 것과 같이 복수의 탐색 중심에서 만들어지는 diamond search 패턴은 탐색 중심의 위치 관계에 따라 생성되는 탐색 지점들이 모두 다르게 나타나기 때문에, 완전한 자유도의 참조 데이터 접근이 요구된다. 때문에, 제안된 IME 하드웨어에서는, 이러한 완전한 자유도의 참조 데이터 접근을 제한하여, 여러 탐색 중심을 동시에 고려하여 처리하는 것이 아닌, 단일 cycle에 하나의 탐색 중심 만을 처리하도록 제한하였다. 이렇게 하면 diamond search 패턴의 탐색 중심을 참조 데이터 버퍼의 중앙에 오도록 참조 데이터를 fetch 할 수 있다. 또한, SAD-tree를 참조 데이터 버퍼의 중앙을 탐색 지점으로 하여 고정된 주변의 픽셀 영역과 직접 연결할 수 있다. 따라서, 참조 데이터 버퍼와 SAD-tree 사이에는 참조 데이터의 위치를 선택하기 위한 Mux로직이 불필요 하게 된다. 그러나, TZS에는 diamond search 이외에도 sampled raster search 패턴이 존재하며, 8x8 CU가 아닌 다른 CU를 위해 탐색 범위가 1인 raster search가 수행되어야 하므로, 적어도 각 32개의 SAD tree는 세 가지의 search 패턴을 지원할 수 있는 3 to 1 Mux를 반드시 사용해야한다. 그림 4.8에 24x24 픽셀 참조 데이터 버퍼에 대하여, 접근 가능한 탐색 지점을 각 search 패턴 별로 나타내었다. 이와 같이 자유도를 엄격히 제안하면 참조 데이터 버퍼와 SAD-tree 사이에 필요한 Mux는 단지 40.1 k gate만 필요하며, 적은 양의 로직만으로도 TZS를 32개의 SAD-tree에 서로 다른 탐색 지점을 위한 참조 데이터를 공급할 수 있다.



(a)



(b)



(c)

그림 4.9 24x24 픽셀 크기의 참조 데이터 버퍼에 SAD-tree가 접근 할 수 있는 세가지 패턴: (a) diamond, (b) sampled raster, (c) raster

4.5 하드웨어 구조

4.5.1 전체 하드웨어 구조

제안된 하드웨어 기반의 IME의 하드웨어 구조를 그림 4.10에 나타내었다. 제안된 하드웨어는 5-스테이지로 구성되어 있다. ‘A’ 스테이지에는 4.3.3절에 설명한 컨텍스트 스위칭을 위한 스케줄러와 참조 데이터 fetch를 위한 주소 생성기가 있다. ‘F’ 스테이지에는 4.3.3절에 설명한 multi bank 구조의 참조 데이터 SRAM이 있으며, 16x16 원본 데이터를 저장하고 있는 버퍼가 존재한다. ‘C’ 스테이지에는 8개의 16x16 SAD tree와 32개의 8x8 CU의 MV 중 더 나은 16개의 탐색 지점을 선택하기 위한 비교기가 있다. ‘b’ 스테이지에는 32x32, 64x64 CU의 SAD를 생성하기위한 축적기가 있으며, 최적 MV를 결정하기위한 비교기가 있다. ‘B’ 스테이지에서는 각 참조 픽쳐, PU, CU의 최적 모드 결정을 위한 비교기가 있다.

그림 4.11에는 asymmetric motion partition (AMP)를 지원할 수 있는 16x16 SAD tree의 구조를 나타내었다. 기존의 SAD-tree는 정사각형 블록 형태의 중간 SAD값만을 구할 수 있기 때문에, 기존 SAD-tree 구조에 AMP의 SAD를 계산할 수 있도록 구현되었다. 16x16 CU의 AMP의 SAD 값은 4x4 SAD값으로부터 합산되어 구해지는 것을 볼 수 있다.

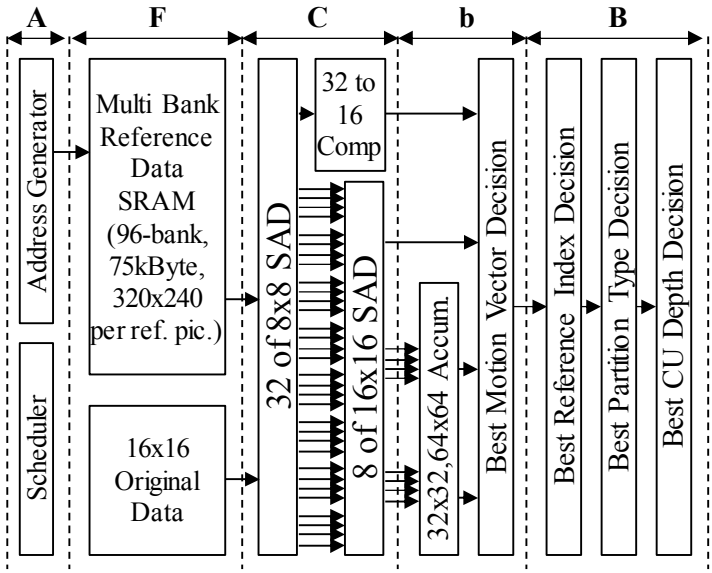


그림 4.10 제안된 하드웨어 기반 IME의 하드웨어 구조

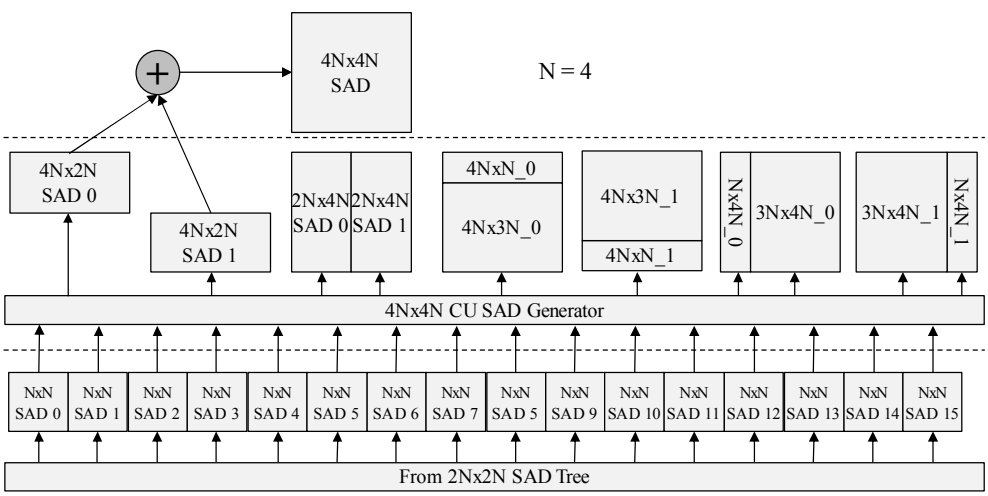


그림 4.11 AMP의 SAD를 구하기 위한 16x16 SAD tree의 구조

그림 4.12에는 참조 데이터 메모리와 탐색 가능 영역을 나타내었다. CTU를 기준으로 탐색 범위는 (257,177) 픽셀의 직사각형 형태이며, 가장 큰 크기의 CU인 64x64 CU를 고려하였을 때, 필요한 참조 데이터 메모리의 크기는 (320,240) 픽셀 크기이다. 참조 데이터 메모리 내의 현재 CTU의 기준 위치는 (128,88) 지점이며 좌측으로 129 픽셀, 우측으로 128픽셀, 상단으로 89픽셀, 하단으로 88픽셀의 만큼의 탐색이 가능하다.

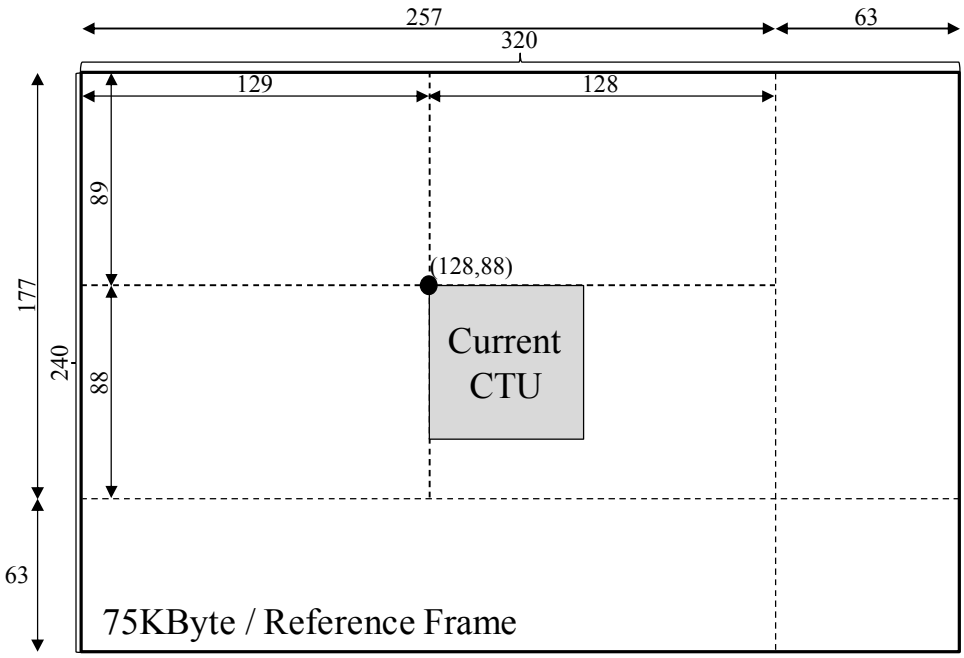


그림 4.12 제안된 하드웨어 기반 IME를 위한 참조 데이터 메모리와 탐색 가능 영역

그림 4.13에는 제안된 multi bank 메모리 구조로부터 참조 데이터를 fetch 하는 예를 나타내었다. 이 예에서는 (94,93) 위치의 24x24 픽셀 크기의 참조 데이터를 fetch하기위한 과정을 나타낸다. 320x240 픽셀 크기의 참조 데이터 메모리는 총 100개의 주소를 갖는 96개의 64bit SRAM으로 이루어져 있다. 흰색 사각형으로 분할된 참조 데이터 영역은 100개의 주소를 나타낸다. 참조 데이터가 96개의 SRAM에 저장된 62번 주소의 예를 들어 음영 처리되지 않은 굵은 사각형으로 표시하였다. 즉, 62번 주소 영역의 참조 데이터를 모두 fetch 하려면 96개의 SRAM 모두의 62번 주소에 접근하면 한 번에 62번 주소 영역에 해당하는 32x24 픽셀 영역을 fetch 할 수 있다. 만약 회색 음영 처리된 사각형 위치인 (94,93) 위치의 참조 데이터를 fetch 하고자 한다면 SRAM에 12, 13, 22, 23의 주소를 요청된 참조 데이터 영역에 맞도록 적절하게 할당해야 한다.

그림 4.14에 그림 4.13에 나타낸 (94,93) 영역을 fetch 할 때 필요한 SRAM bank와 해당 bank의 주소를 나타내었다. 그림에서 A로 표시된 영역은 주소 23에 해당하는 영역이고, B로 표시된 영역은 주소 12에 해당하는 영역이고, C로 표시된 영역은 주소 13에 해당하는 영역이며, D로 표시된 영역은 주소 22에 해당하는 영역이다. B 영역을 fetch 하기 위해 SRAM bank 79, 83, 87, 91, 95에는 12번 주소가 요청 되어야 하며, C 영역을 fetch 하기 위해 SRAM bank 76, 77, 78, 80, 81, 82, 84, 85, 86, 88, 89, 90, 92, 93, 94에는 13번 주소가 요청 되어야 한다. 나머지 영역 A와 D에 마찬가지로 각 bank에 22와 23번 주소가 요청되어야 한다.

96개의 bank에 적절히 주소값을 할당한 뒤 SRAM bank 0부터 bank 95까지의 출력 데이터를 순서대로 배열한 뒤 byte align에 맞지 않는 데이터를 제외하고 나면, 그림 4.15 (a)와 같이 데이터가 나타난다. 그러나, 실제로 요청된 참조 데이터의 배열은 그림 4.15 (b)와 같다.

그러므로 최종적으로 실제로 요청된 참조 데이터를 fetch하는 절차를 완료하기 위해서는 가로 및 세로 방향으로 방향으로 0~23픽셀의 순환 시프터가 필요하다. 그림 4.13에서 시작한 (94,93) 위치의 참조 데이터를 fetch하는 예시를 위해서는 시프터를 사용하여 좌측으로 2 픽셀, 하단으로 5픽셀을 시프트 하면 최종적으로 요청된 참조 데이터의 형태를 완성할 수 있다.

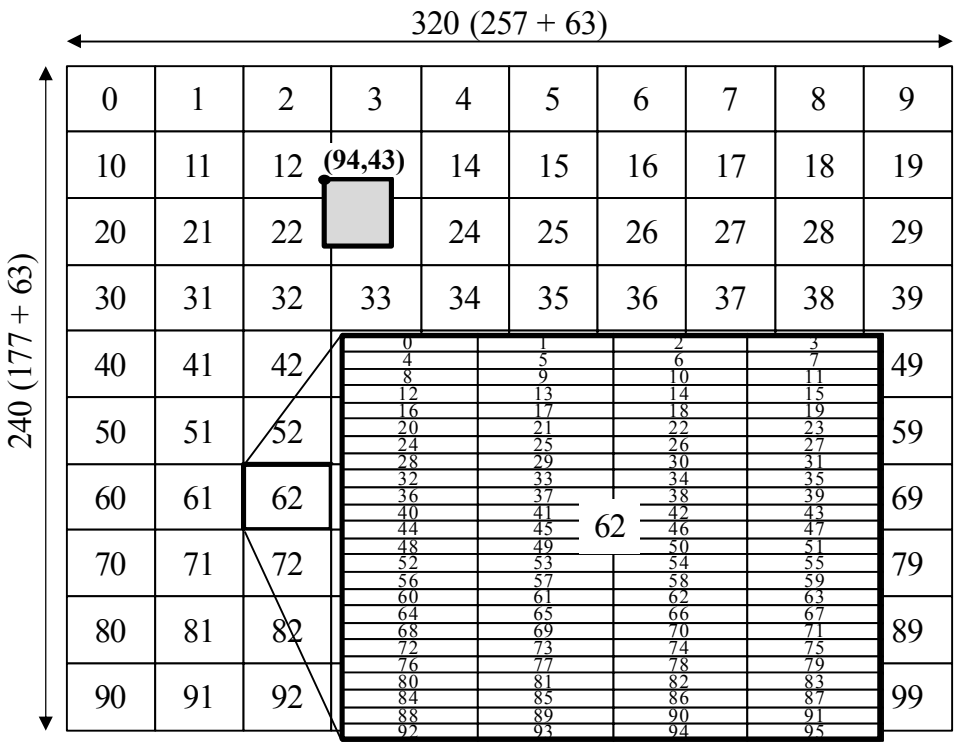


그림 4.13 참조 데이터가 Multi bank에 저장되는 주소 및 배치 구조

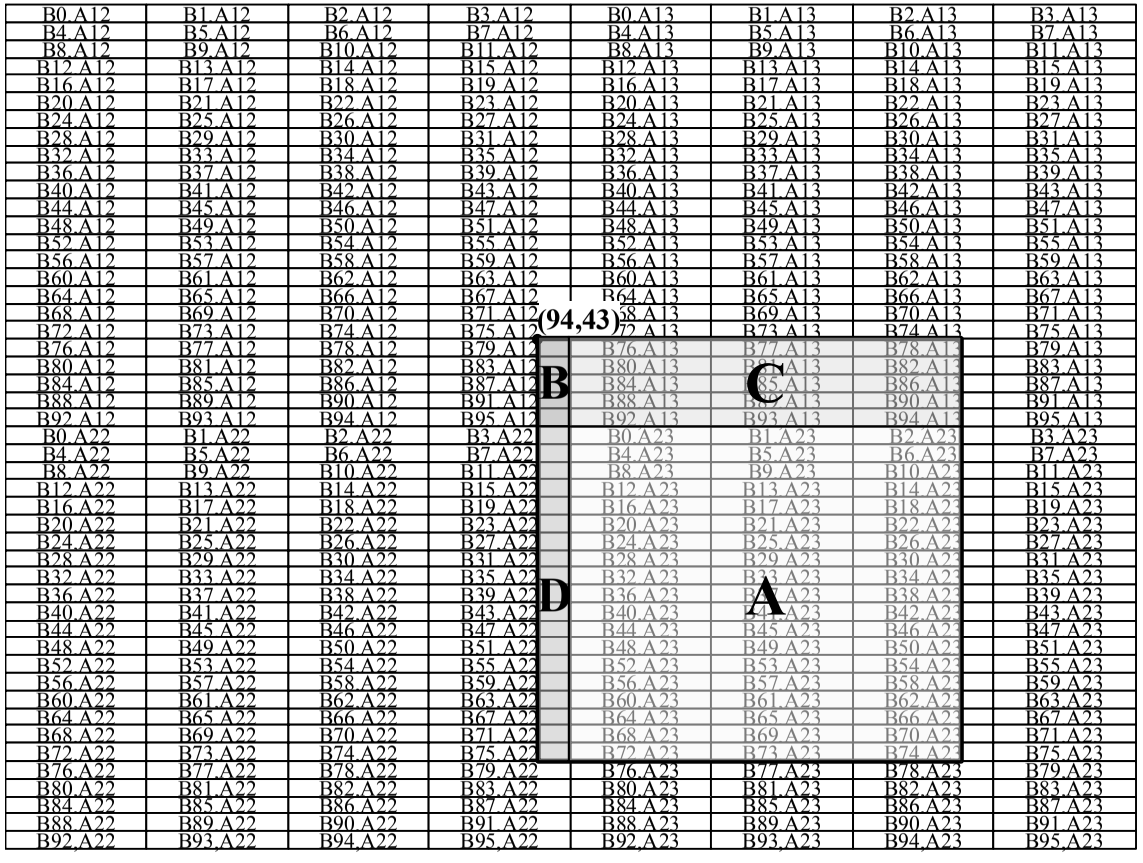
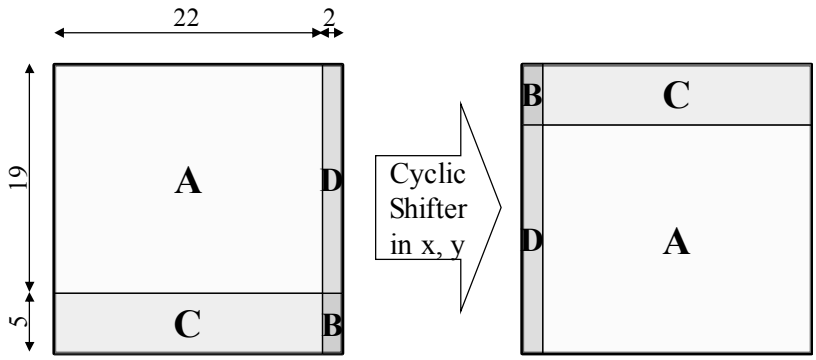


그림 4.14 그림 4.13의 (94,93)위치의 24x24 픽셀 영역을 fetch 할 때 사용되는 SRAM bank와 주소
(B: bank number, A: address)



(a) (b)

그림 4.15 (a) 그림 4.14에서 Multi bank SRAM으로부터 fetch된 데이터, (b) 그림 4.13에서 실제 요청된 참조 데이터

4.5.2 하드웨어 세부 스케줄

이 절에서는 하드웨어의 세부 스케줄에 대해서 설명한다. 우선 그림 4.16에 Scheduler의 동작을 pseudo-code 형태로 나타내었다. 이 동작은 매 cycle 마다 반복되며, 앞서 4.3.3절에서 설명한 컨텍스트를 참조하여 수행된다. 먼저 8x8 CU를 위한 컨텍스트를 먼저 체크한다. 참조 픽쳐를 0 부터 3 까지 반복하여 해당 참조 픽쳐에 idle 상태가 아닌 PU 스레드가 존재하는지를 확인한다. Idle 상태가 아닌 PU 스레드가 존재하는 경우 이 스레드를 실행할 수 있도록 컨트롤러를 조절하고 스케줄러는 동작을 종료한다. 만약 모든 PU의 스레드가 idle 상태인 경우 다음 참조 픽쳐로 넘어가 동일한 동작을 수행한다. 만약 8x8 CU에 대해 모든 참조 픽쳐에서 idle 상태가 아닌 PU 스레드가 발견되지 않은 경우 depth 2, 1, 0 의 순서로 각 depth내의 참조 픽쳐를 순환하며 BMVP가 유도 되었는지를 확인한다. 하위 depth로부터 BMVP가 유도된

상위 depth가 있는 경우 작은 CU를 우선순위로 하여 IME가 실행 될 수 있도록 스케줄 한다. 만약 모든 상위 depth의 CU에 대해서도 수행 가능한 스레드가 존재하지 않는 경우, 현재 cycle은 idle cycle이 된다.

```
Algorithm IME Scheduler
Input: Context for 8x8 CU, ctx8; Context for none 8x8 CU, ctxn8
Output: Is idle cycle, idle; Allowed CU depth, depth; Allowed
reference index, refidx
1:  for RefIdx=0 to 3 do
2:    if ctx8[RefIdx].istherevalidPUthread=true then
3:      idle = false
4:      depth = 3
5:      refidx = RefIdx
6:      return idle, depth, refidx
7:    end if
8:  end for
9:  for Depth=2 to 0 do
10:   for RefIdx=0 to 3 do
11:     if ctxn8[RefIdx].istherevalidBMVP=true then
12:       idle = false
13:       depth = Depth
14:       refidx = RefIdx
15:       return idle, depth, refidx
16:     end if
17:   end for
18: end for
19: idle = true
20: return idle
```

그림 4.16 하드웨어 기반의 IME를 위한 스케줄링 알고리즘

그림 4.17에는 제안된 하드웨어 기반 IME의 세부 파이프라인 스케줄의 예를 나타내었다. 상단에 표시된 숫자는 cycle을 나타내며 A, F, C, b, B는 다섯 개의 파이프라인 스테이지를 나타낸다. 아래 쪽에는 8x8 CU의 2Nx2N, 2NxN_0, 2NxN_1, Nx2N_0, Nx2N_1 및 16x16 CU, 32x32 CU의

동작 상태를 나타내었다. 첫 번째 cycle에는 8x8 CU의 2Nx2N PU의 스레드가 초기 diamond search를 수행하고 이어서 8x8 CU의 2Nx2N_0, Nx2N_1 PU의 스레드가 병렬로 초기 diamond search를 수행한다. 이는 두 PU의 탐색 중심이 같기 때문이다. 세 번째 cycle에는 8x8 CU의 2NxN_1, Nx2N_1 PU의 초기 diamond search가 수행된다. 네 번째 cycle에는 모든 PU 스레드의 diamond search가 파이프라인 동작 중 이므로 상위 depth인 16x16 CU의 IME를 파이프라인 지연 시간인 4 cycle동안 수행된다. 이후에 하나의 PU 스레드가 raster search가 필요하게 되어 64 cycle동안 raster search를 수행하게 된다. 72번째 cycle에는 raster search의 파이프라인 시작이 모두 종료되어 다시 4 cycle의 지연이 생기게 되는데 이 때에는 16x16 CU을 위한 BMVP가 존재하지 않아서 32x32 CU를 위한 IME를 수행하게 된다. 77, 77, 78, 79번째 cycle에는 raster search로부터 찾아진 최적의 MV를 탐색 중심으로 하여 8x8 CU의 다섯 개 PU스레드의 동작이 이루어지고 다시, 4 cycle의 지연이 발생하면 32x32 CU의 두 번째 BMVP에 대한 IME를 수행한다. 8x8 CU의 첫 번째 반복 탐색의 파이프라인 지연이 끝나면 84, 85번째 cycle에 8x8 CU의 2Nx2N PU 스레드와 2NX2N_0, Nx2N_0 PU 스레드에 대한 두 번째 반복 탐색이 이루어진다. 이후 다시 idle cycle이 발생하면 32x32 CU의 세 번째 BMVP에 대한 IME 연산이 4 cycle동안 이루어지고 90번째 cycle에는 2Nx2N PU 스레드를 위한 IME가 1 cycle이루어진 다음 하나의 8x8 CU에 대한 동작이 모두 종료된다. 유의할 점은 가독성을 위해서 이 예에서는 한 장의 참조 픽쳐의 사용 만을 예로 들었으나, 다수의 참조 픽쳐가 사용되는 실제 경우에는 8x8 CU의 idle cycle에 상위 depth에 대한 IME 보다 우선적으로 서로 다른 참조 픽쳐에 대한 8x8 CU의 IME가 이루어진다.

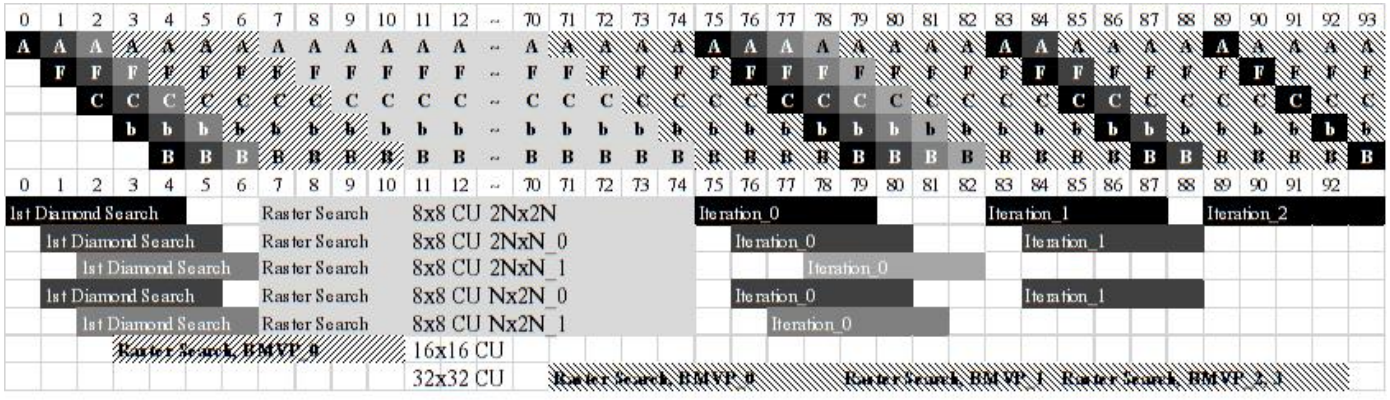


그림 4.17 제안된 하드웨어 기반 IME의 세부 파이프라인 스케줄의 예

4.6 하드웨어 구현 결과 및 실험 결과

4.6.1 하드웨어 구현 결과

이 절에서는 4장에서 제안된 하드웨어 기반의 IME의 구현 결과를 나타낸다. 제안된 하드웨어는 Verilog-HDL로 설계되어 65nm 공정으로 Synopsys Design Compiler로 합성 되었다. 전체 하드웨어 리소스 사용은 1,269 K gates이며 1.2V 동작 전압으로 최대 400 MHz 로 동작 할 수 있다. Multi bank SRAM 참조 데이터 메모리를 구현하기 위해 96 bank의 SRAM을 사용하였으며, 참조 픽쳐 한 장당 320x240 픽셀 크기의 75kBytes만큼의 데이터를 저장한다. 압축 효율 변화 측정을 위해서는 HM 13.0의 LDP 공통 테스트 조건으로 IME를 수행한다. CTU 크기는 64x64 이며 모든 CU 크기와 PU 타입을 지원하며, 4장의 참조 픽쳐가 사용된다. 단, 탐색 범위는 CTU를 기준으로 가로 및 세로 방향으로 ±128, ±88 픽셀의 영역으로 한정되었다. 최대 동작 주파수에서 64x64 CTU 한 개당 3292 cycle budget을 할당할 경우 3840×2160 해상도의 영상을 60 fps의 속도로 IME 수행이 가능하다. 이때의 압축 효율 변화는 HM 13.0 참조 소프트웨어 대비 0.11%의 BD-BR 저하가 나타난다.

표 4.4 제안된 하드웨어 기반 IME의 구현 결과

Supported CU size	64×64, 32×32, 16×16, 8×8
Supported PU Type	2N×2N, 2N×N, N×2N,Nx2N, 2N×nU, 2N×nD, nR×2N, nL×2N
Number of Reference Picture	4
Search Range	257×177 (CTU centered)
Technology	TSMC 65-nm CMOS General Purpose
Supply Voltage	1.2V @ 400MHz
Average Throughput	121,500 64×64CTUs/s
RD Degradation (BD-BR)	0.11%
Operating frequency	3840×2160 60fps @ 400MHz
Gate Count	1,269 K
Internal SRAM	96 bank, 75 Kbyte /reference picture

또한 표 4.5에 제안된 하드웨어의 모듈 별 하드웨어 리소스 사용량을 나타내었다. 제안된 하드웨어는 크게 세 가지의 특징을 갖는다. 32개의 8x8 CU의 탐색 지점을 단일 cycle에 탐색이 가능한 8개의 16x16 SAD tree, idle cycle을 제거하기 위한 컨텍스트 스위칭을 사용한 스케줄링, 넓은 영역의 참조 데이터에 대한 빠른 접근이 그것이다. 이러한 관점에서 세부 모듈 별 하드웨어 리소스를 살펴보면, 탐색 포인트의 병렬도가 32개 포인트로 매우 높기 때문에 SAD tree와 Cost Calculator가 667 K gates로 가장 많은 부분을 차지한다. 또한 컨텍스트 스위칭을 구현하기 위한 컨텍스트 스토리지가 78 K gates의 하드웨어 리소스를 사용한다. 24x24 픽셀 영역의 참조 데이터와 16x16 픽셀 영역의 원본 데이터를 빠르게 공급 할 수 있도록 단일 cycle에 많은 양의 참조 데이터를 처리하기 때문에 이를 위한 버퍼와 스위칭 mux 로직이 303.9 K gates 만큼의 하드웨어 리소스를 사용하였다. 또한 bottom-up MV 예측을 위해

하위 depth의 MV를 저장하기 위한 버퍼로 57.1 K gates의 하드웨어 리소스가 사용되었다.

표 4.5 제안된 하드웨어 세부 모듈 별 하드웨어 리소스 사용 내역

Component	Gate Count(K)
Context based Scheduler	93.2
SAD tree & Cost Calculator	667.0
Best SP Decision	36.0
Best Ref., PU, CU Decision	34.1
Ref., Org. Data Buffer & Feeder	303.9
Context Storage	78.0
BMVP Buffer	57.1
Total	1,269.3

4.6.2 수행 시간 및 압축 효율

이 절에서는 제안된 하드웨어 기반 IME의 수행 시간 및 압축 효율을 나타낸다. 표 4.6에 제안된 하드웨어로 IME를 수행한 17개 영상에 대한 압축 효율 및 수행 cycle을 나타내었다. 제안된 bottom-up IME는 최하위 depth에서만 수정된 TZS 알고리즘을 적용하고 나머지 상위 depth에 대해서는 탐색 범위가 1인 raster search를 수행한다. HM과 비교하였을 때 압축 효율의 변화는 BD-BR 로 0.00%로 HM과 동일한 압축 효율 성능을 가진다. 단, 이 때의 CTU당 IME 수행 시간은 영상 평균 3543 cycles로 3840x2160 영상을 60fps로 수행하기 위해 필요한 3292 cycle에 미치지 못한다. 컨텍스트 스위칭을 사용하여 idle cycle을 제거하였기 때문에 평균 idle cycle은 전체 수행 시간 중 4.31%에 불과하다. 영상 별로 수행

시간을 자세히 살펴보면, 영상내 움직임 특성이 복잡하고 빠른 영상인 PeopleOnStreet, BasketballDrive, Kimono, RaceHorses, RaceHorsesD의 영상에서 각각 5394, 5722, 6676, 7231, 6089 cycle이 소요되는 것을 볼 수 있다. 반면, 움직임이 단순한 FourPeople, Johnny, KristenAndSara의 영상에 대해서는 984, 1001, 1234 cycle로 매우 짧은 수행 시간을 가지는 것을 볼 수 있다. 이는 제안된 IME 하드웨어에 적용된 수정된 TZS알고리즘의 수행시간이 움직임이 복잡한 영상에 대해서는 수행 시간이 크게 증가하는 특성을 가지기 때문이다. 예를 들어, raster search가 필요하지 않은 PU의 경우, 다섯 개 PU의 AMVP가 모두 일치하는 경우에는 단 1 cycle에 한 장의 참조 픽쳐에 대한 IME가 종료될 수 있지만, raster search가 일단 한 번 이라도 수행되면 최소 64 cycle의 수행 시간이 요구되기 때문이다.

게다가, 표 4.6에 나타낸 하드웨어의 수행 시간은 많은 CTU에 대한 IME를 수행하여 측정된 평균 값으로, CTU단위로 살펴보면 기준 수행 시간인 3292 cycle을 초과하는 CTU는 매우 많이 존재할 것이다. 그러므로 CTU 단위로 엄격하게 3292 cycle의 수행시간을 준수할 필요가 있는 시스템에서는 기준 수행 시간에 맞추어 CTU에 대한 IME를 끝낼 수 있도록 조절해야 한다. 수행 시간을 제어하기 위해서 제안된 IME의 스케줄러에 수행 시간 제어를 위한 알고리즘을 추가하였다. 이 알고리즘은 주어진 CTU당 수행 시간을 8x8 CU의 개수인 64개로 나누어 각 CU에 할당한 뒤, TZS를 수행 하면서 할당된 수행 시간이 모두 사용되면 TZS를 조기 종료 하도록 한다. 단, 여러 장의 참조 픽쳐가 존재하므로, 현재 픽쳐와 가장 가까운 참조 픽쳐를 우선한다. 이는 현재 픽처와 시간적 거리가 가까울수록 높은 확률로 최적의 MV가 발견되기 때문이다. 또한, TZS의 raster search를 위한 탐색 범위를 CU에 할당된

수행 시간에 따라 제한하였다. 마지막으로, 앞서 수행된 CU 에서 수행 시간이 남는 경우 이후에 실행되는 CU에 이월하여 할당한다. 설명된 방법으로 하드웨어 수행 시간을 3292 cycle로 모든 CTU에 대해 엄격하게 제한한 경우의 제안된 하드웨어의 압축 효율과 수행 시간을 표 4.7에 나타내었다. 영상 평균 BD-BR은 0.11% 증가로 매우 적은 압축 효율 저하가 나타났다. 평균 수행 시간은 3292 cycle로 상한선이 결정되었으므로, 3292 cycle보다 훨씬 낮은 1652 cycle로 나타났다. 기존의 cycle수에 제한이 없는 경우 긴 수행 시간을 필요로 했던 PeopleOnStreet, BasketballDrive, Kimono, RaceHorses, RaceHorsesD 영상의 수행 cycle은 2000 cycle 이상의 큰 값을 보인다. 또한 이 영상들에 대해서는 많은 수의 CTU에서 TZS가 마지막까지 수행되지 못하였으므로 압축 효율의 저하가 다른 영상보다 크게 나타나야 한다. 각 영상 별로 0.59%, 0.65%, 0.22%, 0.07%, -0.07%의 BD-BR 변화를 보인다. 이는 기존 수행 시간 제한이 없는 경우보다 크게 나타난 것이다. 다른 영상들에 대해서는 비교적 적은 BD-BR 증가를 보이는데, 이는 3292 cycle의 수행 시간 제한 전에 CTU에 대한 IME가 모두 종료되는 CTU가 많았기 때문이다.

표 4.6 제안된 하드웨어 기반 IME의 압축 효율 및 수행 시간

Video Sequence		BD-BR(%)	Processing Cycle for a CTU		
			IME Cycle	Idle Cycle	Total
ClassA	Traffic	-0.04	1582	85	1667
	PeopleOnStreet	0.02	5277	117	5394
ClassB	BasketballDrive	0.41	5330	392	5722
	BQTerrace	-0.05	2064	134	2198
	Cactus	0.02	2994	136	3130
	Kimono	0.00	6460	216	6676
	ParkScene	0.04	2421	134	2555
ClassC	BQMall	-0.01	3402	168	3570
	BasketballDrill	0.06	3400	174	3575
	PartyScene	-0.01	3164	137	3330
	RaceHorses	-0.14	6983	247	7231
ClassD	BasketballPass	-0.19	2913	161	3074
	BlowingBubbles	-0.10	2650	164	2814
	RaceHorses	-0.23	5878	211	6089
ClassE	FourPeople	0.05	915	69	984
	Johnny	0.23	934	68	1001
	KristenAndSara	-0.02	1165	69	1234
	Average	0.00	3384	158	3542

표 4.7 CTU당 수행 시간을 3292 cycle로 엄격하게 제한한 제안된 하드웨어 기반 IME의 압축 효율 및 수행 시간

Video Sequence		BD-BR(%)	Processing Cycle for a CTU		
			IME Cycle	Idle Cycle	Total
ClassA	Traffic	0.00	1073	74	1146
	PeopleOnStreet	0.59	2063	119	2182
ClassB	BasketballDrive	0.65	2105	408	2513
	BQTerrace	-0.06	1335	104	1439
	Cactus	0.13	1302	135	1437
	Kimono	0.22	2332	223	2555
	ParkScene	-0.05	1488	120	1608
ClassC	BQMall	0.11	1462	166	1628
	BasketballDrill	0.24	1367	181	1548
	PartyScene	-0.03	1395	109	1505
	RaceHorses	0.07	2430	257	2687
ClassD	BasketballPass	0.07	1327	152	1479
	BlowingBubbles	-0.14	1392	148	1540
	RaceHorses	-0.07	2215	207	1809
ClassE	FourPeople	0.16	646	66	711
	Johnny	0.05	738	62	800
	KristenAndSara	-0.05	830	63	893
	Average	0.11	1500	153	1652

4.6.3 제안 방법 적용 단계 별 성능 변화

이 절에서는 4장에서 설명된 제안 방법들을 순차적으로 적용 하였을 때 하드웨어의 수행 cycle과 BD-BR 관점에서 압축 효율의 변화에 대해서 자세히 살펴본다. 그림 4.18에 제안된 방법들을 HM 참조 소프트웨어로부터 시작하여 순차적으로 적용하였을 때 CTU당 평균 수행 cycle과 평균 압축 효율을 나타내었다. 짙은 회색 막대는 non-idle cycle을 나타내며, 옅은 회색 막대는 idle-cycle을 나타낸다. 흰색 바탕의 숫자는 cycle 타입 별 cycle의 수를 나타내고, 굵게 표시된 숫자는 HM 참조 소프트웨어 대비 압축 효율의 변화를 BD-BR로 나타낸 것이다. 하드웨어 cycle은 본 연구에서 제안한 cycle 당 32개 8x8 PU 탐색점을 처리 가능한 하드웨어를 기준으로 하였다.

방법 1은 HM의 TZS을 제안된 하드웨어 구조에 그대로 적용 하였을 경우를 나타낸다. 즉, PU 단위로 TZS를 순차적으로 수행하는 경우를 나타낸다. 방법 1의 경우 HM 보다 압축 효율이 0.21%p 만큼 향상 된 것을 볼 수 있다. 이 압축 효율의 향상은 다음의 설명에 기인한다. TZS에서 diamond search를 수행 할 때, 탐색점의 거리를 1부터 두 배씩 증가하는 거리로 순차적으로 확장하여 탐색을 수행하다가, 한차례 확장에도 불구하고 더 나은 MV가 발견되지 않는 경우 조기 종료 한다. 반면, 단일 cycle에 거리 8까지 diamond search의 모든 탐색점을 한 번에 탐색하는 제안된 하드웨어에서는 조기 종료 없이 넓은 범위를 탐색 하여, 지엽적인 극솟값에 빠져 잘못 된 최적 MV를 선택하는 경우를 막을 수 있다. 또한 TZS의 raster search 에서는 5x5 픽셀 영역 당 1개의 탐색점만을 탐색하고, 제안된 하드웨어에서는 4x4 픽셀 영역 당 2 개의 탐색점을 탐색 하기 때문에 마찬가지로 지엽적인 극솟값에 빠지는

경우를 막을 수 있다. 그러므로, TZS를 제안된 하드웨어에서 수행하는 경우 0.21%p의 압축 효율 향상 효과를 가진다. 다음으로 방법 2는 방법 1에 더하여 4.2.1절에서 설명한 PU 별 TZS 수행 루프를 CU 단위로 풀어서 PU의 TZS 스레드들이 동시에 수행 될 수 있도록 한 경우를 나타낸다. PU별 TZS 수행 루프를 풀어냄으로써, 각 PU는 다른 PU의 TZS가 종료되기를 기다리지 않고도 TZS의 수행이 가능하게 되어, 서로 다른 PU의 idle cycle 동안에 탐색을 수행할 수 있게 된다. 그러므로 idle cycle이 감소하게 되어 23178 cycle에서 3728 cycle로 수행 cycle이 감소하게 된다. 단, 이 때 4.2.1절에서 설명한 바와 같이, 각 PU 타입의 두 번째 PU의 AMVP가 정확하게 유도되지 못하므로 이로 인한 압축 효율의 저하가 0.20%p만큼 나타나, 결과적으로 -0.01%의 압축 효율 변화를 보인다. 방법 2에서는 idle cycle만 감소하므로 방법 1 과 비교하였을 때, non-idle cycle이 변화하지 않을 것으로 생각할 수 있지만, AMVP의 유도 결과가 방법 1과 방법 2가 다르기 때문에, TZS의 수행이 서로 달라져 non-idle cycle의 수행 cycle 또한 달라지게 되는 점에 유의하라. 다음으로 방법 3은 방법 2에 추가적으로 4.2.3에서 설명한 각 PU들의 중복 연산 감소 방법을 적용한 결과를 나타낸다. 단일 CU내 최대 13개의 PU들간의 중복 연산이 매우 높은 확률로 나타나므로, non-idle cycle이 99107 cycle에서 22167 cycle로 약 78% 감소하게 된다. 단지 중복 연산만을 제거하였을 뿐 이므로, idle-cycle과 압축 효율에는 아무런 변화가 나타나지 않는다. 다음으로 방법 4는 방법 3에 추가적으로 각 PU들의 TZS로 발생된 탐색점을 모든 PU가 공유하도록 한 경우를 나타낸다. 이 경우, 각각의 PU들이 갖는 고유한 TZS 스레드가 발생시키는 탐색점이외에도 다른 PU가 발생시킨 탐색점에서도 서로의 최적 MV를 탐색 할 수 있도록 하였기 때문에, 단일 PU의 TZS 스레드에

의해 발생한 탐색점에서만 탐색을 수행하는 것 보다 훨씬 많은 탐색점에서 탐색을 수행할 수 있는 효과를 갖게 된다. 때문에 0.29%p만큼의 압축 효율이 향상되어, 결과적으로 -0.30%의 압축 효율 향상 효과를 갖게 된다. 수행 cycle의 경우 22167 cycle에서 21056 cycle로 약간 감소하게 되었는데, 이는 PU들이 서로의 탐색점을 공유하여 한 번의 반복 과정에서 보다 많은 탐색 점을 탐색 할 수 있게 되면서 약간의 반복 횟수 감소 효과를 나타냈기 때문이다. 다음으로 방법 5는 방법 4에 더하여 3장에서 제안한 bottom-up IME방법을 적용한 경우를 나타낸다. Bottom-up IME에 의해 16x16, 32x32, 64x64 CU depth에 대한 IME는 탐색 범위 1로 제한 되었으므로, IME의 연산량이 감소하여 non-idle cycle이 21056 cycle에서 3384 cycle로 감소하게 되었다. 또한, 상위 depth의 CU에 대해서는 TZS가 적용되지 않고 raster search가 적용되었기 때문에, 상위 depth CU의 TZS에서 발생하던 idle-cycle이 제거되어 전체 idle-cycle의 감소 효과를 보였다. 압축 효율 측면에서는 상위 depth의 탐색 범위를 1로 제한하였기 때문에 방법 4 대비 0.30%p의 압축 효율 저하를 보여 결과적으로 0.00%로 HM 참조 소프트웨어와 동일한 압축 효율을 보였다. 마지막으로 본 연구에서 제안한 방법들을 모두 적용한 방법 6은 방법 5에 더하여 서로 다른 참조 픽처와 서로 다른 CU의 컨텍스트 스위칭을 통해 TZS의 idle cycle을 제거한 결과이다. 이 경우 방법5에서 단지 idle-cycle만이 감소하게 되므로 non-idle cycle에는 변화가 없다. 또한 압축 효율에서 변화는 나타나지 않는다. 그러므로 HM의 TZS를 그대로 하드웨어로 수행한 방법 1의 대비 본 연구에서 제안한 방법 6은 단지 약 3%의 수행 시간 만으로 HM 참조 소프트웨어와 동일한 압축 효율 성능을 보이는 하드웨어 기반 IME를 달성 할 수 있다.

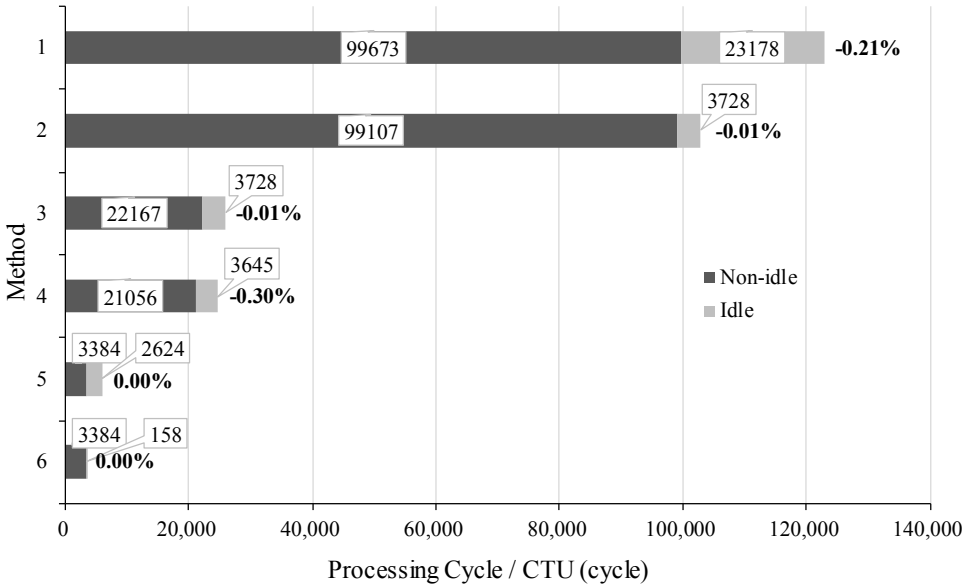


그림 4.18 제안 방법들을 순차적으로 적용 하였을 때 하드웨어의 수행 cycle과 압축 효율의 변화

4.6.4 이전 연구와의 비교

마지막으로 과거에 연구되었던 하드웨어 기반 IME와 비교를 수행하여 표 4.8에 나타내었다. [34]는 64x64, 32x32, 16x16 PU만을 각 PU에 대해 별도의 IME 엔진을 할당하여 세 개 depth의 PU에 대한 IME를 동시에 진행하였다. 또한 TZS의 단계별 의존성 특성을 회피하기 위해 diamond search를 3-step search로 대체하였으며, 움직임이 큰 PU를 위해서 sampled raster search를 3-step search와 병렬로 수행하도록 하였다. 이렇게 하면 움직임이 작은 경우와 움직임이 큰 경우에 모두 대응할 수 있지만, 결과적으로 한 쪽 IME 엔진에서의 탐색 결과가 버려지게 되므로 비효율성을 야기한다. [35]는 기존의 TZS알고리즘으로부터

출발하여 하드웨어에 맞게 각 단계를 수정 및 삭제하였다. 참조 데이터의 reuse를 위해서 diamond search 패턴을 변형하여 사방으로 퍼지는 형태의 패턴을 3회 초과 반복 수행부터 최적 MV의 위치가 진행하는 방향에 대해서만 탐색을 수행하도록 수정 하였다. 그러나, 사실상 16x16 PU를 위해서만 탐색 범위를 16으로 하고 나머지 PU들은 탐색 범위를 1 또는 4로 제한하였기 때문에 기존 diamond search를 이용한 반복 탐색이 갖는 의미가 퇴색되었다. 게다가 raster search를 제거하고 오직 diamond search 만으로 IME를 수행하도록 하였다. 그러므로 이 과거 연구는 실험에 따라 공통 테스트 영상에 대해서만 잘 동작 할 수 있도록 세밀하게 조정된 하드웨어이기 때문에 범용성이 결여되었다. [36]은 본 논문과 유사하게 TZS알고리즘을 변형 없이 그대로 사용하였으나, 1 cycle에 8x8 PU의 한 개의 탐색 지점을 탐색 할 수 있는 throughput을 가진 하드웨어로 탐색을 수행하며, 8x8 PU 한 개에 대해 92 cycle의 수행 시간을 정적으로 할당하였지만, TZS의 단계 별 의존성 문제에 대한 해결책은 제시하지 않고있다. 그러므로 TZS의 단계 별 의존성과 참조 데이터 fetch에 필요한 최소 파이프라인인 3-stage 파이프라인을 고려하면 실질적으로 하드웨어 사용율은 33% 선에 머무를 것이다. 즉, 조기 종료되는 경우 TZS 탐색을 한다고 볼 수 있지만, 움직임이 많은 영상에 대해서는 채 한 번의 반복 수행하기 어려워, TZS의 제대로 된 성능을 전혀 발휘 할 수 없다. [37]의 경우 가장 적은 하드웨어 리소스를 가지지만 오직 16x16 PU에 대해서만 IME를 수행한다. 이 연구는 고속화를 위해 diamond search를 다섯 개의 탐색 중심에서 동시에 수행하도록 하여 병렬도를 높였다. 그러나 이 연구 역시 diamond search의 탐색 중심 업데이트에 대한 idle cycle에 대한 해결책을 제시하지 못하였다. [38]의 경우 가장 직관적인 구현으로써, 127x127의 탐색 영역을

64x64 CU크기의 SAD tree로 HEVC가 가지는 모든 PU 크기에 대해 full raster search를 수행하도록 한 연구이다. 이 연구는 모든 PU 크기를 지원하기 때문에 우리 연구의 접근 방식과 비슷하지만, 고속 알고리즘을 사용하지 않았기 때문에, 3.65M gate라는 본 논문의 IME 하드웨어의 세 배에 가까운 하드웨어 리소스를 사용하였다. 결과적으로 본 논문의 IME 하드웨어는 1.27 M gate의 하드웨어 리소스를 사용하지만, 한 장의 참조 픽처를 사용하는 다른 과거 연구와 대조적으로 4장의 참조 픽쳐를 지원한다. 또한 1개 ~ 8개의 PU 타입만을 선택적으로 수행하는 것과 대조적으로 24개의 PU 타입을 실시간으로 처리할 수 있다. 또한 TZS 알고리즘을 압축 효율이 개선 되도록 수정하여, PU단위 AMVP의 부재 및 bottom-up IME에 의한 압축 효율의 손실을 보상하였다. 그 결과, HM 대비 화질 저하가 0.11%로 거의 없기 때문에 HM의 인코딩 성능을 그대로 실시간으로 하드웨어로 구현이 가능함을 보였다는데 의의가 있다.

표 4.8 제안된 하드웨어 기반 IME와 과거 연구와의 비교

	[34]	[35]	[36]	[37]	[38]	Proposed
Number of Ref. Pic.	1	1	1	1	1	4
Number of PU Type	3	8	3	1	24	24
Search Algorithm	3-step, Sampled Raster	Simplified TZ Search	Simplified TZ Search	MPDS (Diamond)	Full Raster Search	Modified TZ Search
Throughput(Resolution)	3840x2160 @30fps	4096x2048 @60fps	3840x2160 @30fps	1920x1080 @30fps	3840x2160 @30fps	3840x2160 @60fps
*Normalized Throughput (kCTU/sec)	182.25	983.04	182.25	15.19	1,458	11,664
Normalized Throughput per kGate	0.33	5.37	1.18	0.30	0.40	9.18
Gate Count	551K	183K	154K	51K	3.65M	1.27M
***BD-BR	1.6%	3.1%	2.45%	**1.7%	Not Avail	0.11%

*Throughput in kCTU × NumRef × NumPU, **Bit-rate, ***참조 픽쳐 및 PU 생략으로 인한 BD-BR 변화는 반영되지 않은 값

다음으로 그림 4.19에 Gate Count에 따른 Normalized Throughput을 과거 연구들과 비교하여 나타내었다. 제안된 하드웨어는 1.27M gate에 11664 kCTU/sec의 throughput을 가져, gate count 측면에서는 [38]과 나머지 과거연구들의 중간에 위치한다. 과거 연구들 중에서 [38]은 3.65M gate의 가장 많은 하드웨어 리소스를 소모하여 과거 연구 중에서는 가장 높은 1458 kCTU/sec의 throughput을 나타내었으나, 제안된 하드웨어의 throughput의 12.5%에 해당하는 것이다. 나머지 연구들은 500 K gate 대 이하의 적은 하드웨어 리소스를 사용하였지만, 그 throughput이 제안된 하드웨어의 1.3% ~ 8.4%에 불과하다. 이러한 큰 차이는 다음에 기인한다. 과거 연구에서는 사용되지 않았던 고성능의 고속 IME 알고리즘인 TZS의 사용이 움직임이 단순한 영상에 대한 수행 속도를 대폭 상승 시켰다. TZS의 사용을 위해서 하드웨어의 idle cycle을 줄이기 위한 노력으로 PU의 루프를 풀어 수행하도록 하였고, 서로 다른 참조 픽처와 CU의 컨텍스트 스위칭을 사용하였다. 또한 중복 연산 제거를 통해 약 76%의 연산량을 감소 시켰다. 그리고 여기에 추가적으로 bottom-up IME를 사용함으로써, 제안된 하드웨어는 PU들 간의 중복 연산 제거와 bottom-up IME를 통해 약 97%의 연산량을 압축 효율 손실 없이 줄였기 때문에 이와 같은 큰 차이가 나타나는 것이다.

그림 4.20에서는 KGate 당 Throughput기여도 및 압축 효율을 과거 연구들과 비교하여 나타내었다. 어두운 회색 점선은 과거 연구들의 추세를 나타내며, 연한 회색 점선은 제안된 하드웨어를 두 가지의 throughput으로 동작 시켰을 때를 나타낸다. 8k-UHD 30fps 의 경우 4k-UHD 60fps 보다 cycle budget이 절반이기 때문에, 압축 효율의 저하가 심화되지만 다른 연구들이 throughput을 증가시키기 위해 저하되는 압축 효율의 손실에 비해 월등히 적은 양임을 알 수 있다.

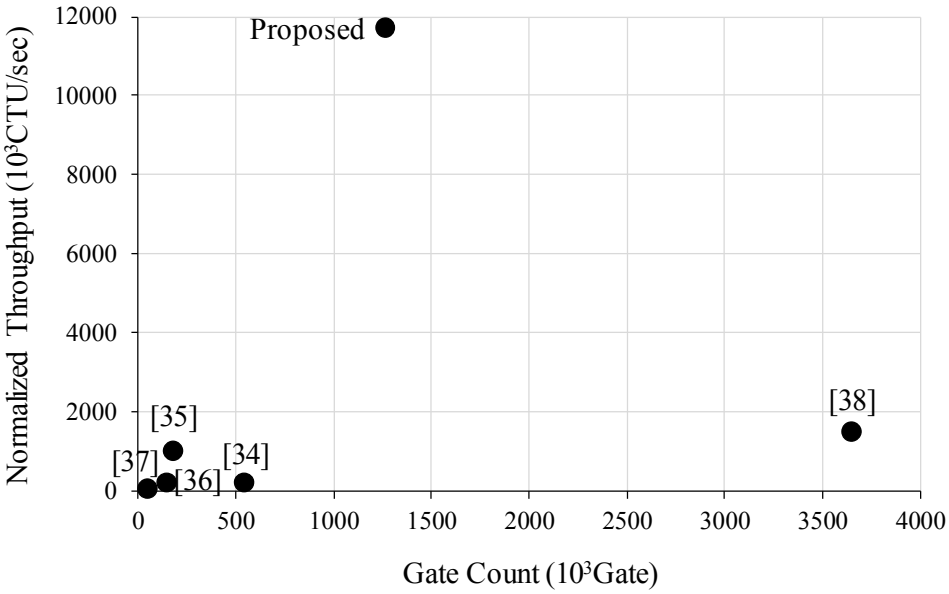


그림 4.19 제안 방법과 과거 연구들의 하드웨어 리소스 사용량 및 정규화된 CTU 처리량 비교

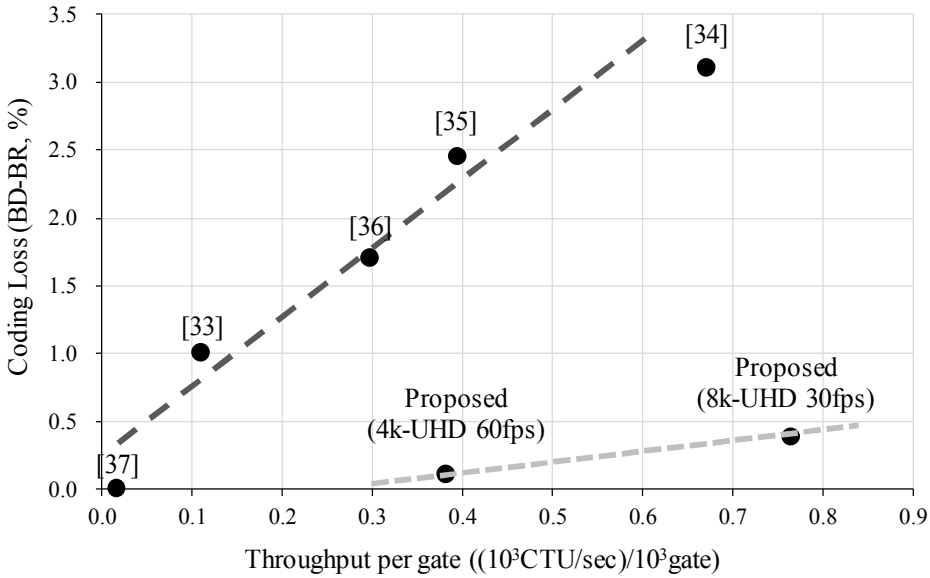


그림 4.20 제안 방법과 과거 연구들의 KGate 당 Throughput 기여도 및 및 압축 효율 비교

제 5 장 하드웨어 기반 Merge Mode Estimation

5.1 기존 Merge Mode Estimation의 하드웨어 관점에서의 고찰

5.1.1 기존 Merge Mode Estimation

이 절에서는 기존 MME가 갖는 계산 복잡도의 변동 특성에 대해서 다룬다. 그림 5.1에 MME의 순서도를 나타내었다. MME는 candidate construction, motion compensation (MC), best mode decision 단계로 이루어져 있다. 각 단계는 NumMergeCands로 표시되는 merge 후보의 수만큼 반복하여 동작된다. 먼저 candidate construction 과정에서는 현재 PU와 merge 될 후보의 리스트를 구성하고 이웃 PU의 움직임 정보를 저장한다. 다음으로 MC 과정에서는 이웃 PU의 움직임 정보가 가리키는 참조 데이터를 fetch 하여 필요에 따라 보간 필터를 적용한다. 만약, 이웃 PU의 움직임 정보가 bi-prediction인 경우 각 참조 픽쳐 리스트 별로 보간 필터가 적용되어야 한다. MVx와 MVy는 가로 및 세로 방향의 MV 성분을 나타낸다. 그림의 IsFracMV는 merge 후보의 MVx와 MVy가 소수 단위의 MV를 포함하는지 아닌지를 판단한다. IsFracMV의 결과에 따라 가로방향 및 세로 방향의 보간 필터 적용 여부가 결정된다. 다음으로 best mode decision 단계에서는 SATD 기반의 RD cost가 계산된다. RD cost의 distortion은 SATD로부터 얻어지며, bit는 merge flag및 merge

index로부터 얻어진다.

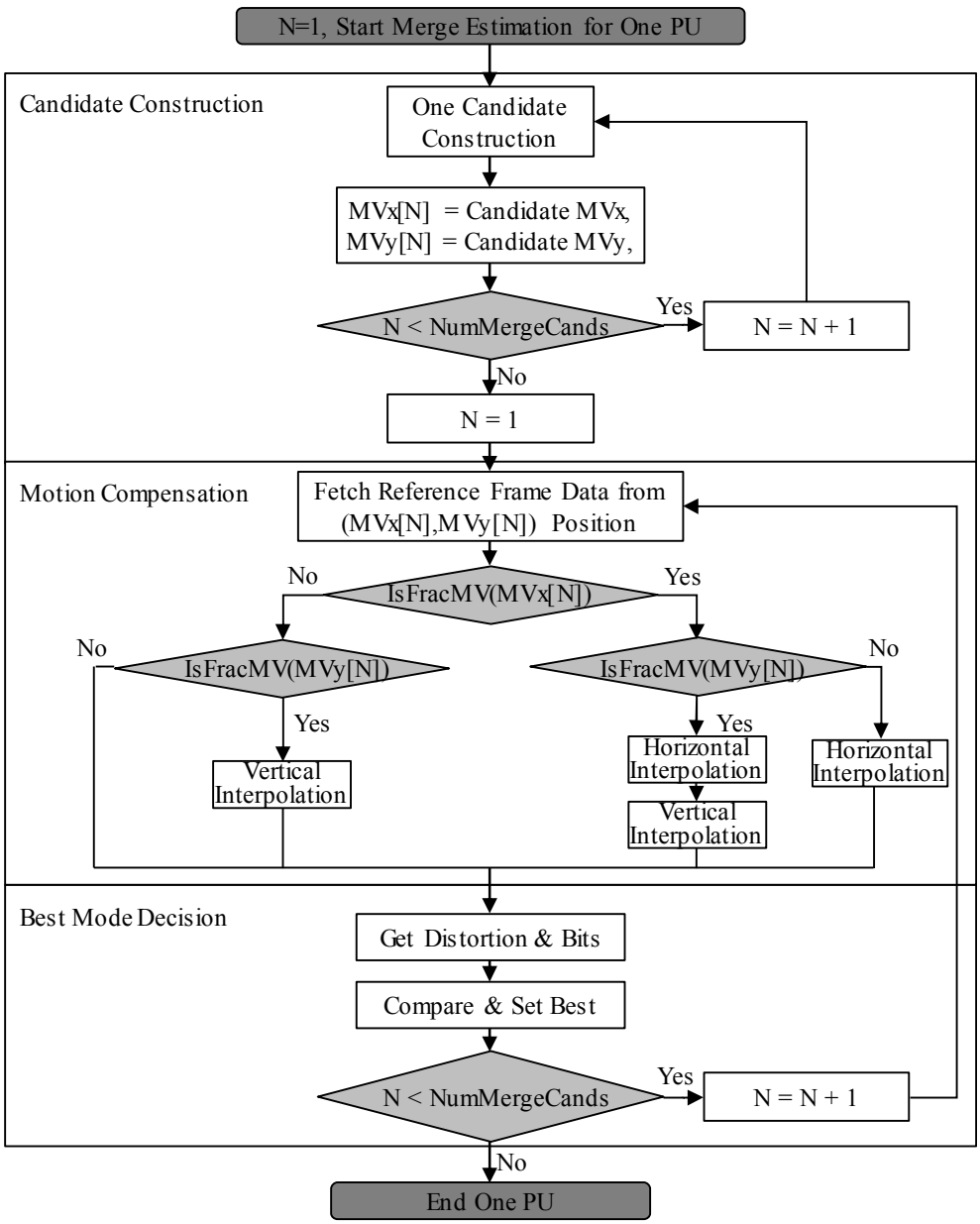


그림 5.1 Merge Mode Estimation의 순서도

5.1.2 기존 Merge Mode Estimation 하드웨어 구조 및 분석

5.1.1절에서 설명한 것과 같이 MME의 다섯 개의 merge 후보는 이웃 PU의 움직임 정보로부터 MV를 얻는다. 이 MV는 이웃 PU의 움직임 정보에 의해 결정되므로 integer MV (IMV), fractional MV (FMV)가 될 수도 있다. 게다가 이웃 PU가 bi-prediction으로 예측 되었다면, 한 개의 MV를 갖는 uni-prediction과 다르게 두 개의 MV를 가지게 된다. 그러므로 계산 복잡도가 전적으로 이웃 PU의 움직임 정보에 의해서 결정되며, 이 계산 복잡도는 MV의 수와 MV가 IMV인지 FMV인지에 따라서도 달라지게 된다. Merge 후보가 uni-prediction이고 IMV를 갖는다면 가장 적은 계산 복잡도를 가지게 되고, 반면 bi-prediction이고 FMV를 갖는다면 가장 높은 계산 복잡도를 가지게 된다.

이러한 특징에 대해 하드웨어 관점에서 더 자세히 분석하기위해 기존 MME를 수행하는 MME 하드웨어의 예를 그림 5.2에 나타내었다. 이 하드웨어는 32-픽셀 병렬도를 가지고 SATD연산을 위해 8x8 SATD를 채택하였음을 가정하였다. 매 cycle 마다 8x4 픽셀 크기의 블록이 파이프라인 방식으로 처리된다. MV를 위해서는 4개의 모듈이 존재한다. 8x4 픽셀 크기의 가로 방향 보간 필터와 8x12 픽셀 크기의 세로 방향 보간 필터 버퍼, 그리고 8x4 픽셀 크기의 세로 방향 보간 필터와 bi-prediction을 위한 8x4 픽셀 크기의 평균 및 차이 연산 모듈이 존재한다. 이 모듈들은 merge 후보의 MV가 가리키는 참조 데이터 영역의 데이터를 fetch 하여 보간 필터가 적용된 참조 데이터를 생성한다. 세로 방향의 보간 필터 버퍼는 세로 방향의 보간 필터를 적용하기위해 8-tap 만큼 가로 방향 보간 필터의 결과를 저장하기 위한 공간이다. 보간 필터가 모두 적용되고 MC가 종료된 픽셀 데이터를 원본 픽셀 데이터와

차이값을 구한 다음에는 8x4 픽셀 크기의 가로 방향 변환이 수행되고 이를 8x8 픽셀 크기의 transpose array에 잠시 저장하였다가 4x8 픽셀 크기의 세로 방향 변환을 수행하여 SATD cost를 계산한다. 이러한 MME 하드웨어의 동작 과정에서 가장 높은 계산 복잡도를 가지는 부분은 보간 필터 부분이다. 보간필터에는 다수의 곱셈 연산이 포함되어 있기 때문이다. 특히 하드웨어 리소스측면에서 세로 방향의 보간 필터는 가장 많은 리소스를 필요로 한다. 왜냐하면 그림 5.2에 나타낸 것과 같이, 세로 방향의 보간 필터를 수행하기 위해서는 가로 방향의 보간 필터가 적용된 픽셀을 저장하기 위한 세로 방향 보간 필터 버퍼가 필요하다. 또한 세로 방향의 보간 필터의 입력 픽셀의 bit 폭은 이미 가로 방향의 보간 필터가 적용된 이후이므로 16bit의 폭을 갖는다. 이 bit 폭은 가로 방향의 보간 필터 입력인 8bit의 두 배 이다. 결과적으로 그림 5.2에 나타낸 하드웨어의 각 모듈 별 하드웨어 리소스 사용량을 보면, 가로 방향 보간 필터는 63,406 gates, 세로 방향 보간 필터는 140,641 gates, SATD 연산 모듈은 79,174 gates의 하드웨어 리소스를 필요로 한다.

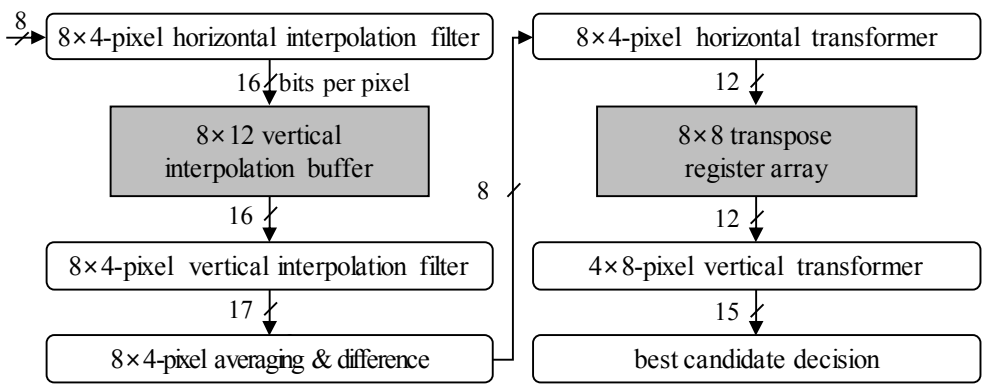
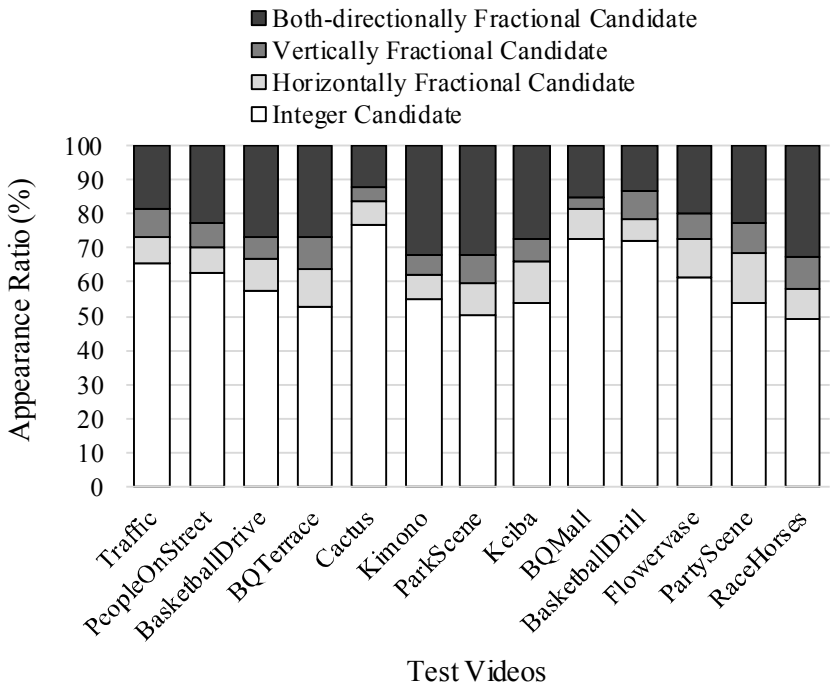


그림 5.2 기존 Merge mode estimation을 위한 하드웨어 구조

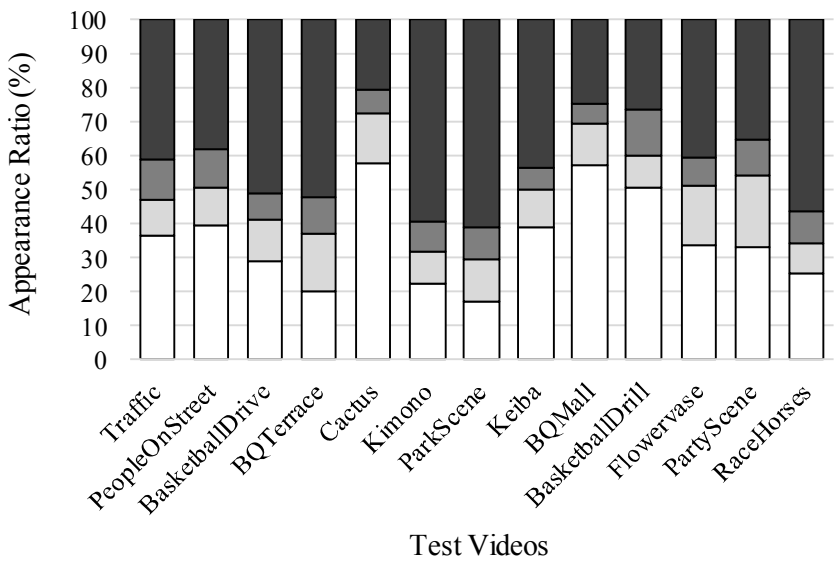
5.1.3 기존 Merge Mode Estimation의 하드웨어 사용률 저하 문제

그림 5.3에 네 가지의 merge 후보 타입의 출현 빈도를 영상에 따라 나타내었다. 이 네 가지의 후보 타입은 다음과 같이 분류된다. 먼저, 가로 및 세로 방향으로 모두 소수 단위 성분의 MV가 존재하지 않는 경우 integer candidate (Icand)라고 칭한다. 다음으로 만약 MV가 가로 방향으로만 소수 단위 성분의 MV가 존재하는 경우, horizontal candidate (Hcand)라고 칭한다. 다음으로 만약 MV가 세로 방향으로만 소수 단위 성분의 MV가 존재하는 경우 vertical candidate (Vcand)라고 칭한다. 마지막으로 가로 및 세로 방향으로 모두 소수 단위 성분의 MV가 존재하는 경우 bi-directional candidate (Bcand)라고 칭한다. 그림 5.3의 가로축은 다양한 테스트 영상을 나타내고, 세로축은 네 가지 후보 타입의 출현 빈도를 나타낸다. LDP 테스트 조건의 결과를 그림 5.3 (a)에 나타내었고 RA 테스트 조건의 결과를 그림 5.3 (b)에 나타내었다. LDP 테스트 조건에서는 Bcand, Vcand, Hcand, Icand의 출현 빈도가 모든 영상에 대해 평균 23.27%, 7.21%, 9.25%, 60.27%를 나타낸다. 또한 RA 테스트 조건에서는 Bcand, Vcand, Hcand, Icand의 출현 빈도가 모든 영상에 대해 평균 42.31%, 9.20%, 12.85%, 35.64%를 나타낸다. 가로 방향의 보간 필터는 Hcand와 Bcand에 대해서 사용되며, 반면에, 세로 방향의 보간 필터는 Vcand와 Bcand에 대해서 사용된다. 그러므로 가로 방향 보간 필터의 사용율은 LDP와 RA에 대해 각각 32.53%와 55.16%이며 세로 방향의 보간 필터의 사용율은 각각 30.48%, 51.51%이다. 많은 수의 merge 후보가 보간 필터를 필요로 하지 않는다. 일반적으로 실시간 하드웨어 기반의 MME를 위해서는, MME 하드웨어는 가장 계산 복잡도가 높은 다섯 개의 Bcand가 출현한 경우에 대응할 수 있도록 설계된다. 그러나

그림 5.3에 나타난 것과 같이 이러한 방식으로 설계된 MME 하드웨어는 하드웨어 리소스의 심각한 낭비가 일어나게 된다. 특히, 세로 방향의 보간 필터를 위한 하드웨어 리소스의 양은 가장 많지만, 그에 비해 사용 빈도는 절반이 채 되지 않는다.



(a)



(b)

그림 5.3 네 가지 merge 후보 타입의 출현 빈도: (a) LDP, (b) RA

5.2 연산량 변동폭을 감소시킨 새로운 Merge Mode Estimation

5.1절에서 살펴본 바와 같이 MME의 계산 복잡도는 소수 단위 성분의 MV가 merge 후보에 포함되어 있는지 여부와 uni- 및 bi-prediction 여부에 따라 변화한다. 이러한 계산 복잡도의 변동은 하드웨어 리소스의 비효율적인 사용을 야기한다. 그러므로 이 절에서는 MME의 계산 복잡도의 변동폭을 감소시킨 새로운 MME를 제안한다. 그림 5.4에 제안된 하드웨어를 위한 효율적인 MME의 순서도를 나타내었다. 그림. 5.1에 나타낸 기존 MME와 같이 candidate construction, MC, best candidate decision단계가 순서대로 이루어진다. 제안된 MME에서는 두개의 새로운 과정이 candidate construction과 MC 사이에 추가되었다. 두 개의 새로운 과정은 candidate selection과 candidate modification이다. Candidate construction이 수행된 다음 세로 방향의 보간 필터를 수행할 merge 후보가 선택된다. 먼저, IsVFracAllowed가 false로 초기외 된다. 이 array는 각 merge 후보에 대해 세로 방향의 보간 필터가 허용될 것인지 허용되지 않을 것인지를 나타낸다. 다음으로 merge 후보 리스트 내의 merge 후보들은 세로 방향 보간 필터가 우선적으로 적용될 순서로 정렬된다. 이 우선순위는 통계적 분석에 의해 구해진 최적 merge 후보로 선택되는 빈도가 높은 순서와 같다 [26], [39]. 이 순서는 그림 2.3에 나타낸 공간적, 시간적으로 인접한 PU 위치의 A1, B1, 시간 후보, combined bi-predictive 후보, zero 후보, B0, B2, A0의 순서를 갖는다. 이 우선순위로 세로 방향 보간 필터가 적용 될 수 있는 기회가 할당되고 IsVFracAllowed(후보 index)의 값이 true로 설정된다. NumVFracMergeCands는 세로 방향 보간 필터를 적용할 merge 후보 수를 나타낸다. 이러한 선택 과정은 다섯

개의 merge 후보에 대해 모두 적용되거나, 또는, 더 이상 세로 방향 보간 필터 적용 횟수가 남지 않았을 때 종료된다. 마지막으로 IsVFracAllowed값이 true 또는 false 값으로 모두 설정되면 candidate modification이 수행된다. Candidate modification에서는 세로 방향의 보간 필터를 적용할 기회를 얻지 못한 merge 후보에 대해 세로 방향의 MV에 대해 가장 가까운 정수 MV로 반올림 연산을 수행한다. 예를 들어 하나의 merge 후보가 (0,3)과 (0.5, 7)의 두개의 MV를 갖는 경우와 다른 하나의 merge 후보가 (1, 5.5)와 (2.5, 0.75)의 두 개의 MV를 갖는 경우를 고려해 보자. 전자의 경우 세로 방향으로 소수 단위 MV를 가지고 있지 않기 때문에 이 merge 후보에는 아무런 수정이 가해지지 않는다. 반면에 후자의 경우, 세로 방향으로 소수 단위 MV를 가지고 있기 때문에 세로 방향 MV에 대해 반올림 연산이 적용된다. 그 결과 두 번째 merge 후보의 MV는 (1, 6)과 (2.5, 1)로 수정된다. Candidate modification 이후에는 MC와 best candidate decision이 수정된 MV를 사용하여 수행된다. 유의할 점은 수정된 merge 후보의 MV는 MME과정에서만 사용되고 영상 복원 과정과 비트스트림 생성 과정에서는 사용되지 않는다. 이러한 merge 후보 수정으로 인해, 부정확한 최적 merge 후보 결정이 일어날 수 있으므로 압축 효율의 손실이 발생하게 된다. 이 때문에, 계산 복잡도 감소와 압축 효율 간의 트레이드-오프 관계가 나타나게 되므로, 세로 방향 보간 필터 연산을 허용할 merge 후보의 수를 제한할 때에는 이 트레이드-오프 관계를 고려하여 세심하게 결정되어야 한다. 이에 대한 자세한 논의는 5.3.2절에 후술된다.

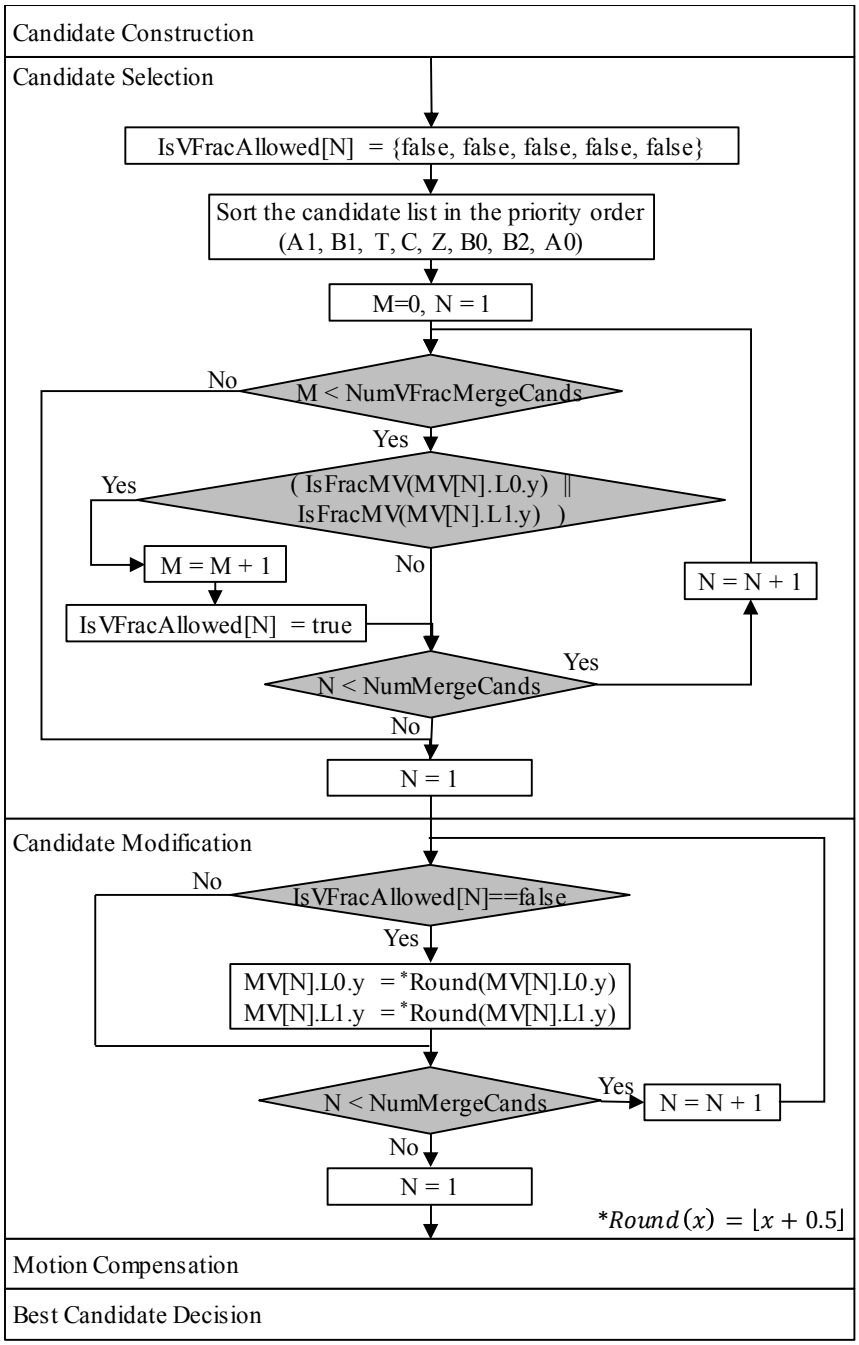


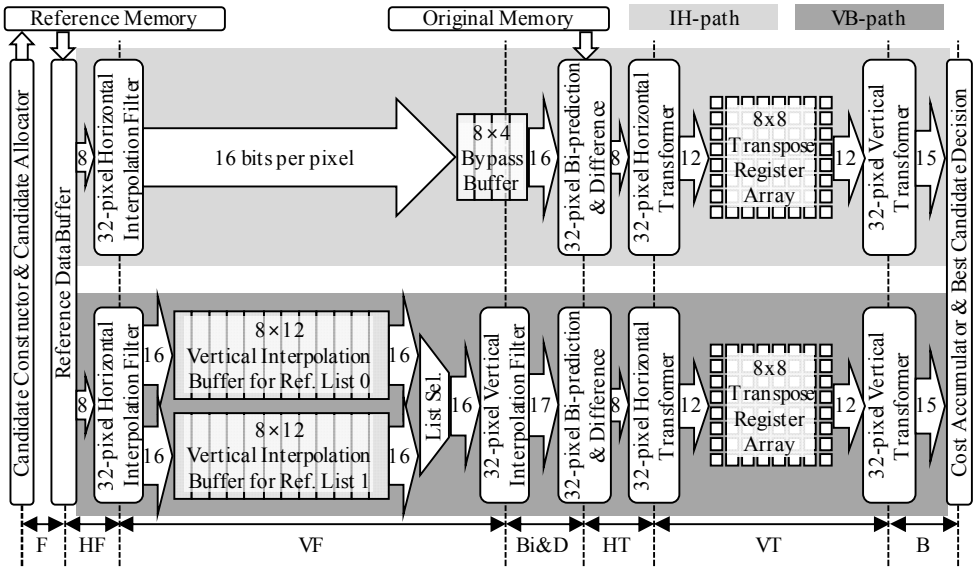
그림 5.4 하드웨어를 위한 효율적인 MME의 순서도

5.3 새로운 Merge Mode Estimation의 하드웨어 구현

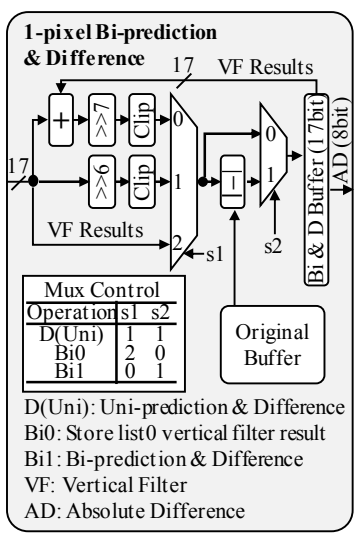
5.3.1 후보 타입 별 독립적 path를 갖는 하드웨어 구조

이 절에서는 제안된 효율적인 MME 하드웨어 구조에 대해 설명한다. 제안된 MME 하드웨어는 merge 후보의 타입에 따라서 병렬 수행되는 두 개의 다른 데이터 패스를 가지는 것이 특징이다. 두 개중 하나의 데이터 패스는 Icand와 Hcand를 위한 것이며, 다른 하나는 Vcand와 Bcand를 위한 것이다. 그림 5.5에 효율적인 MME 하드웨어의 구조를 나타내었다. 그림 5.5 (a)의 연한 회색으로 표시된 패스가 IH-path이며, 이는 Icand와 Hcand를 위한 데이터 패스이다. 어두운 회색으로 표시된 패스가 VB-path이며 이는 Vcand와 Bcand를 위한 데이터 패스이다. IH-path에서 세로 방향 보간 필터의 제거는 앞서 5.1.3절의 소수 단위 MV의 출현이 적은 것을 근거로 하여 이루어졌다. 이 하드웨어 구조에서 만약 IH-path의 수행 시간이 VB-path보다 긴 경우가 발생하면, IH-path의 남은 작업은 VB-path에서 함께 이루어질 수 있다. 하나의 PU에 대한 MME를 시작하기 이전에, candidate constructor와 candidate allocator에서는 merge 후보 리스트가 생성되고, 생성된 merge 후보의 MV의 소수 단위 MV 존재 유무를 확인하여 적응적으로 merge 후보를 각 데이터 패스에 할당한다. 이 적응적 merge 후보 할당 방법은 5.3.2절에 후술 된다. 만약 merge후보가 bi-prediction 인 경우, F (Fetch), HF (Horizontal Filter), VF (Vertical Filter) 스테이지는 cycle 단위로 각 참조 픽처 리스트에 대해 번갈아 가면서 수행되며, Bi & D (Bi-prediction and Difference) 스테이지는 2 cycle에 걸쳐 동작이 이루어져 격 cycle로 결과값을 출력하게 된다. 첫 번째 cycle에서는 한쪽 참조 픽처 리스트의 보간 필터가 이루어진

결과가 버퍼에 저장되며, 두 번째 cycle에서는 저장된 픽셀과 다른 참조 픽처 리스트의 보간 필터가 이루어진 결과를 더하여 bi-prediction을 수행하고 원본 영상과의 차이값을 구한다. 다음으로 8x8 hadamard 변환이 HT (Horizontal Transform)와 VT (Vertical Transform) 스테이지에서 이루어진다. 마지막으로 B (Best) 스테이지에서는 계산된 8x8 SATD cost가 큰 크기의 PU에 대해서 축적되며, 최적 merge 후보 결정이 SATD cost를 이용하여 이루어진다. 그림 5.5 (b)에는 32-픽셀 bi-prediction and difference 모듈의 세부 구조를 나타내었다. Merge 후보가 갖는 MV의 구성에 따라서 2개의 Mux에 의해 동작이 컨트롤 된다.



(a)



(b)

그림 5.5 제안된 효율적인 MME 하드웨어 구조: (a) 전체 하드웨어 구조, (b) 1-픽셀 bi-prediction & difference 구조

5.3.2 하드웨어 사용률을 높이기 위한 적응적 후보 할당 방법

5.3.3절에서 제안된 하드웨어 구조는 IH-path와 VM-path의 두 개의 독립적인 데이터 패스를 갖는다. 만약 하나의 데이터 패스가 먼저 종료되면 다른 데이터 패스가 종료될 때 까지, 먼저 종료된 데이터 패스가 idle 상태가 된다. 그러므로 두 개의 데이터 패스를 모두 사용하여 하드웨어 사용률을 높이기 위해서는 두 개의 데이터 패스에 적절한 로드 밸런싱이 필요하다. 앞서 5.1.3절에서 살펴본 바와 같이 소수 단위 MV를 갖는 merge 후보의 수가 적으므로 두 개의 데이터 패스를 갖는 제안된 하드웨어는 높은 하드웨어 사용률을 위해 적합해 인다. 그러나 실제로, 다섯 개 merge 후보의 MV 구성은 매 PU마다 변화하여 하나의 PU의 다섯 개 merge 후보에 모두 Vcand와 Bcand가 집중되는 현상이 발생할 수 있다. 그림 5.6에 제안된 두 개의 데이터 패스를 갖는 하드웨어의 사용율을 나타내었다. 가로 축은 다양한 테스트 영상을 나타내며 세로축은 하드웨어의 사용률을 나타낸다. LDP와 RA 테스트 조건에서 각각 87.76%와 69.09%의 하드웨어 사용률을 나타낸다. 특히 RA 테스트 조건에서 하드웨어 사용률이 크게 떨어지는데 이는 LDP보다 merge 후보에 포함된 MV의 수가 많기 때문이다. 그러므로 두 데이터 패스의 밸런스를 맞추기 위해서 매 PU 마다 세로 방향의 보간 필터가 허용될 merge 후보의 수의 조절이 적응적으로 이루어져야 한다. 이 때 세로 보간 필터가 허용되는 merge 후보의 수는 하드웨어 사용률 뿐만 아니라, 그에 따른 압축 효율의 저하까지 고려하여 적절하게 결정되어야 한다. 세로 보간 필터가 허용되는 merge 후보의 수를 과하게 제한하는 경우에는 압축 효율의 손실이 커질 것이며, 지나치게 느슨히 제한하는 경우에는 낮은 하드웨어 사용률이 나타날 것이다.

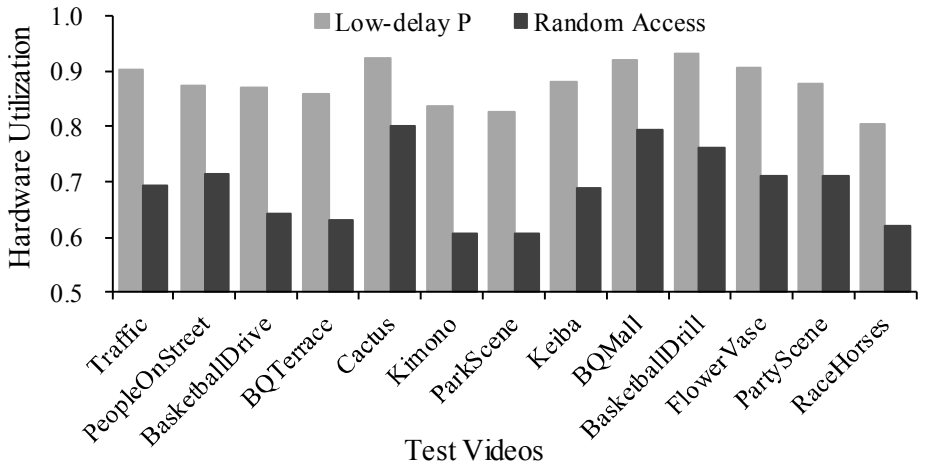


그림 5.6 제안된 효율적인 MME 하드웨어 구조의 하드웨어 사용률

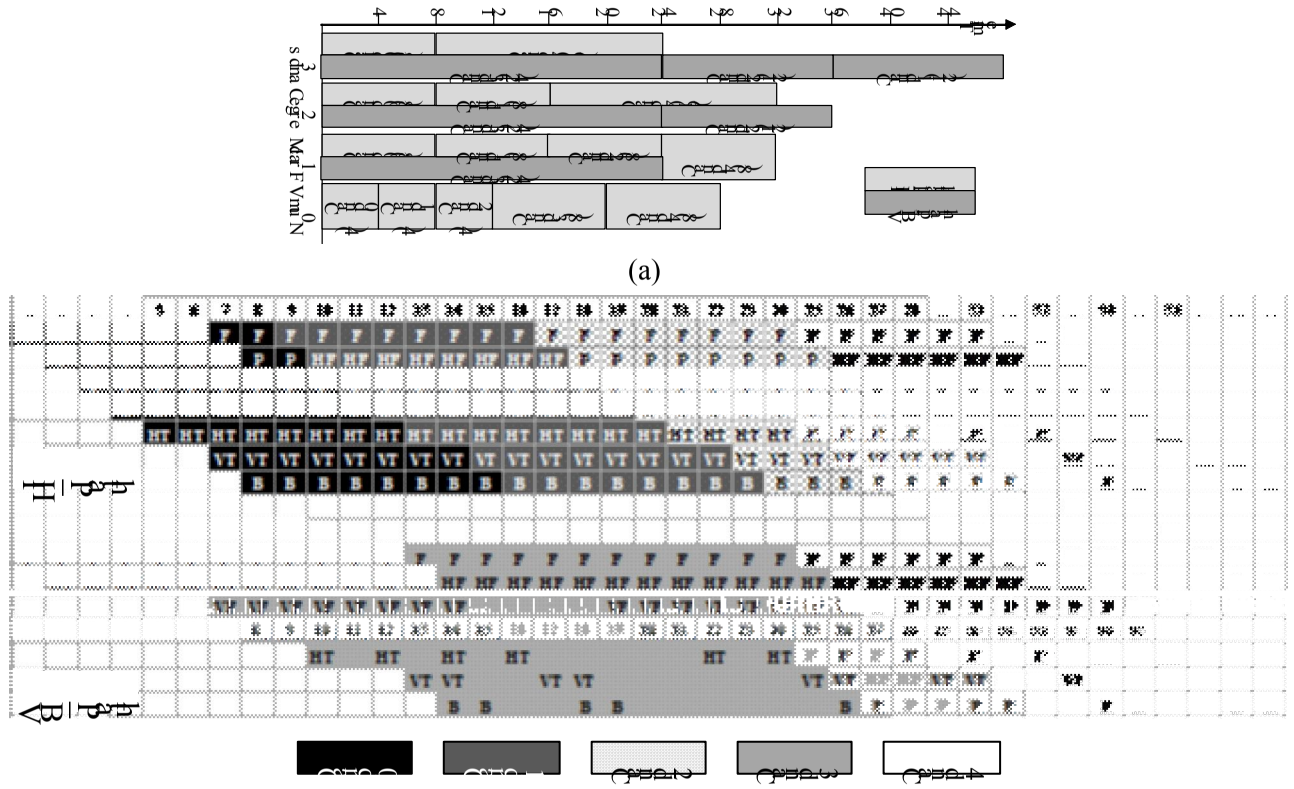
높은 하드웨어 사용률을 유지하면서도 최대의 압축 효율을 얻기 위해서 본 논문에서는 적응적 후보 할당 방법을 제안되었다. 제안된 방법은 다음의 순서로 이루어진다. 먼저 MME를 수행하기 이전에 유도된 각 merge 후보 별 수행 cycle을 예측한다. 다음으로 세로 방향의 보간 필터가 필요한 merge 후보를 VB-path에 한 개씩 할당한다. 이 때 모든 가능한 세로 방향 보간 필터 적용 merge 후보의 수를 0부터 5까지 증가시킨다. 이 때 VB-path에 할당하는 순서는 5.2절에서 나타낸 우선순위를 따른다. 다음으로 VB-path에 할당하고 남은 나머지 merge 후보를 IH-path에 할당한다. 세로 방향 보간 필터를 적용하는 merge 후보의 수의 모든 가능한 경우를 구성한 다음 마지막으로 다음의 두 가지 조건에 따라 최적의 세로 방향 보간 필터 적용 merge 후보의 수를

결정한다. 첫 번째 조건은 높은 하드웨어 사용률이다, 두 번째 조건은 가능한 한 많은 세로 방향 보간 필터 merge 후보의 수이다. 결과적으로 제안된 후보 할당 방법은 full 하드웨어 사용률을 유지하면서도 압축 효율의 손실을 최소화 할 수 있도록 한다.

5.3.3 적응적 후보 할당 방법을 적용한 하드웨어 스케줄

그림 5.7에 제안하는 적응적 후보 할당 방법을 적용한 16x16 PU를 위한 예시를 나타내었다. Cand0, Cand1, Cand2, Cand3, Cand4 의 다섯 개의 merge 후보가 있으며, Cand0, Cand1, Cand2는 uni-prediction이며, 각각 Icand, Bcand, Vcand 이다. 반면에 Cand3와 Cand4는 bi-prediction이며, 각각 Bcand, Hcand이다. Cand3의 두 개의 MV는 모두 소수 단위의 MV를 갖는다. Cand4는 두 개의 MV 모두 가로 방향으로만 소수 단위의 MV를 갖는다. 그림 5.7 (a)의 세로축은 5.2절에 설명한 NumVFracMergeCands를 나타내며, 가로 축은 수행 cycle을 나타낸다. 연한 회색 및 어두운 회색 직사각형은 각각 IH-path와 VB-path의 수행 cycle을 나타낸다. 또한 괄호 안의 숫자는 각 merge 후보에 대한 MME를 수행할 때 소요되는 cycle을 나타낸다. 제안된 MME 하드웨어는 각 데이터 패스에서 32-픽셀을 단일 cycle에 처리 할 수 있도록 설계되었기 때문에, 세로 방향의 보간 필터가 필요한 merge 후보는 12 cycle이 소요되며, 세로 방향의 보간 필터가 필요하지 않은 merge 후보는 8 cycle이 소요된다. 그림에서 NumVFracMergeCands의 값이 3,2,1,0으로 변화 할 때 다섯 개의 merge 후보 모두를 처리하기 위한 시간은 48, 36, 32, 28 cycle이 소요된다. 이 예에서는 적응적 후보 할당 방법에 의해 NumVFracMergeCands의 값은 1로 설정된다. VB-path의 하드웨어 리소스는 Cand3가 종료된 다음 idle

상태가 된다. 그런데 이 때 리소스 공유 방법을 통해서 Cand4는 IH-path와 VB-path의 모든 하드웨어 리소스를 사용해서 두 배의 수행속도인 64 픽셀/cycle의 속도로 수행될 수 있다. 이렇게 하면 idle 상태가 되는 하드웨어가 없이 높은 사용률을 유지할 수 있다. 그림 5.7 (b)에는 NumVFracMergeCands의 값이 1인 경우의 세부 파이프라인 스케줄을 나타내었다. IH-path와 VB-path에 대해 각각 나타내었으며, 가로 축은 수행 cycle을 나타내고, 세로 축은 파이프라인 스테이지를 나타낸다. 각 파이프라인 스테이지의 이름과 동작에 대한 설명은 5.3.1절에 서술되었다.



(a)

(b)

그림 5.7 제안된 적응적 후보 할당 방법을 적용한 16x16 PU를 위한MME 하드웨어 스케줄의 예: (a) NumVFracMergeCands = 3,2,1,0인 경우의 각각 스케줄, 세 개의 uni-prediction인 Icand, Bcand, Vcand와 두 개의 bi-prediction인 Bcand와 Hcand, (b) NumVFracMergeCands = 1 인 경우 세부 파이프라인 스케줄

5.4 실험 결과 및 하드웨어 구현 결과

5.4.1 수행 시간 및 압축 효율 변화

표 5.1에 제안된 하드웨어 기반 MME의 성능 실험 결과를 나타내었다. 첫 번째 열과 두 번째 열은 영상 크기와 영상의 이름을 나타낸다. 세 번째 열과 여섯 번째 열에 나타낸 NumVFracMergeCands값은 VM-path를 통해 세로 방향 보간 필터가 적용된 merge 후보의 평균 수를 나타낸다. 나머지 열에서는 LDP와 RA의 테스트 조건에 대해 압축 효율의 변화를 BD-BR 로 나타내었고 한 개의 64x64 CTU를 처리하는데 소요되는 평균 cycle을 나타내었다. LDP 테스트 조건에 대해서는 모든 영상에 대해 평균 1.058 개의 merge 후보에 대해 세로 방향 보간 필터가 적용 되었고 이 때의 압축 효율 변화는 0.34% 수행 시간은 3,508 cycle이다. RA 테스트 조건에서는 평균 1.163 개의 merge 후보에 대해 의 세로 방향 보간 필터가 이루어졌으며 압축 효율 변화는 0.36%이고 수행 시간은 6,441 cycle이다.

그림 5.8에는 적응적 후보 할당 방법을 적용한 경우와 고정된 수의 후보 할당 방법을 적용한 경우의 성능을 비교하여 나타내었다. 가로축은 허용된 세로 방향 보간 필터가 적용되는 merge 후보의 수를 나타내고 좌측 세로축은 압축 효율의 변화, 우측 세로축은 하드웨어 사용률의 변화를 나타낸다. 제안된 후보 할당 방법은 적응적으로 후보를 할당하기 때문에 그 수가 일정하지 않지만 평균 1~2 구간 사이에서 나타난다. 그림으로부터 제안된 MME 하드웨어는 최대의 하드웨어 사용률을 가지면서도 가장 준수한 압축 효율을 나타내는 것을 볼 수 있다.

표 5.1 제안된 하드웨어 기반 MME의 성능 실험 결과

Video Sequence		Low-delay P			Random Access		
		NumVFrac MergeCands	BD-BR(%)	CTU Cycles	NumVFrac MergeCands	BD-BR(%)	CTU Cycles
ClassA	Traffic	0.989	0.58	3492	1.292	0.62	6617
	PeopleOnStreet	1.046	0.27	3506	1.211	0.31	6187
ClassB	BasketballDrive	1.172	0.17	3542	1.287	0.35	6527
	BQTerrace	1.222	0.39	3554	1.310	0.61	6780
	Cactus	0.570	0.39	3368	0.652	0.27	6286
	Kimono	1.300	0.03	3577	1.421	0.27	6645
	ParkScene	1.321	0.46	3581	1.457	0.43	6716
ClassC	Keiba	1.200	0.20	3549	1.185	0.21	6061
	BQMall	0.664	0.50	3394	0.712	0.34	6222
	BasketballDrill	0.828	0.29	3445	1.054	0.34	6319
	Flowervase	1.013	0.64	3496	1.137	0.52	6629
	PartyScene	1.081	0.28	3514	1.007	0.24	6495
	RaceHorses	1.343	0.29	3586	1.396	0.10	6247
	Average	1.058	0.34	3508	1.163	0.36	6441

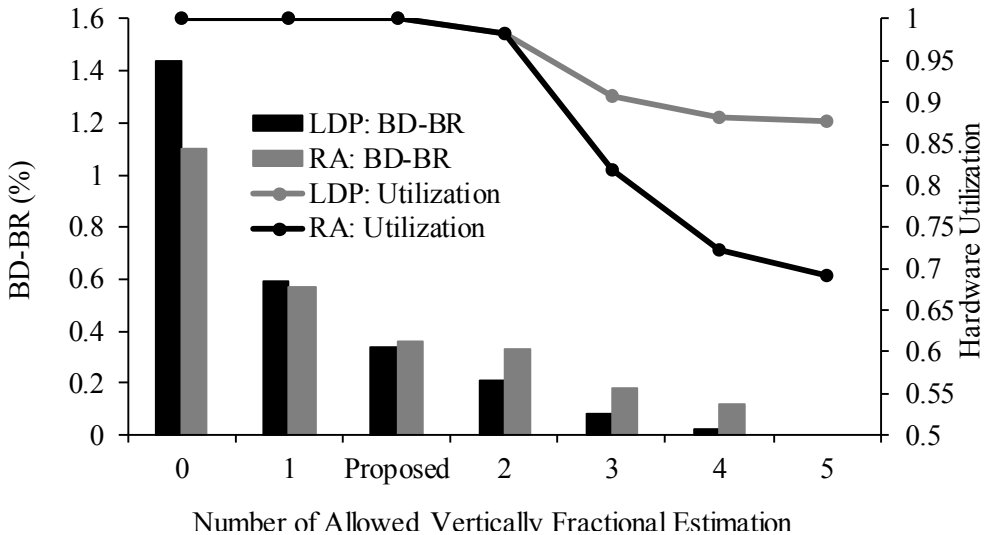


그림 5.8 적응적 후보 할당 방법을 적용한 경우와 고정된 수의 후보 할당 방법을 적용한 경우의 성능 비교

5.4.2 하드웨어 구현 결과

표 5.2에 제안된 하드웨어 기반 MME의 하드웨어 구현 결과를 나타내었다. 제안된 MME 하드웨어는 Verilog-HDL로 구현되었으며 시놉시스 디자인 컴파일러를 사용하여 TSMC 65nm 공정 라이브러리로 합성되었다. 최대 지원 가능한 CTU 크기는 64x64 이며, 8x8 CU를 제외한 모든 CU 크기에서 2Nx2N, 2NxN, Nx2N PU를 지원한다. P-slice와 B0slice를 모두 대응 가능하며, 사용 가능한 최대 merge 후보의 수는 다섯 개이다. 제안된 적응적 후보 할당 방법을 사용하였으므로, 압축 효율의 변화는 P-slice 및 B-slice에 대하여 0.34% 및 0.36%이다. 참조 데이터 공급에 지연이 없는 경우, P-slice 및 B-slice에 대하여 64x64 CTU를 초당 114,025개 또는 62,106개를 처리 할 수 있으며, 이는 400MHz 동작 주파수에서 4k-UHD 영상을 30fps로 처리할 수 있는 속도에 해당한다. 사용된 하드웨어 리소스는 460.8K gate이다.

표 5.2 제안된 하드웨어 기반 MME의 하드웨어 구현 결과

Max CTU size	64×64
Supported PU Type	2N×2N, 2N×N, N×2N (2N×2N only for 8x8 CU)
Slice Type	P-slice, B-slice
Max. Number of Candidate	5
Candidate Type	Spatial, Temporal, Additional
RD degradation	BD-BR P-slice: 0.34% / B-slice : 0.36%
Technology	TSMC 65-nm CMOS General Purpose
Supply Voltage	1.2V @ 400MHz
Throughput @ Max. Freq.	P-slice:114,025 B-slice:62,106 64×64CTUs/s
Operating Frequency	1920×1080 60fps P-slice @ 106.5MHz 1920×1080 60fps B-slice @ 195.6MHz 3840×2160 30fps P-slice @ 213.1MHz 3840×2160 30fps B-slice @ 400.0MHz
Gate Count	460.8K

제 6 장 Overall Inter Prediction

6.1 CTU 단위의 3-stage 파이프라인 Inter Prediction

이 절에서는 3,4,5장 에서 제안된 하드웨어 기반의 IME와 MME가 HEVC를 위한 하드웨어 기반 inter prediction 하드웨어 내에서 함께 동작할 때 고려되어야 할 사항에 대해서 다룬다. 본 논문에서 제안된 IME와 MME 하드웨어는 CTU 단위의 파이프라인 스케줄로 동작 할 수 있도록 설계되었다. Inter prediction을 위한 파이프라인 스케줄은 IME, FME, MME, mode decision (MD) 의 네 가지 작업을 몇 개의 스테이지에 적절히 배분 하느냐에 따라 달라질 수 있다. 일반적으로 과거 H.264/AVC및 다른 연구에서도 널리 사용되어 온 구조는 3-stage의 파이프라인 스케줄로, IME/FME/MD를 각 스테이지에 할당하였다. CTU 단위의 파이프라인 스케줄을 사용하면 MVP유도를 위한 이웃 PU의 정확한 움직임 정보를 파악 할 수 없기 때문에, RD cost의 bit 항의 정확도가 떨어질 수 있다. 그러나, 이로 인한 압축 효율의 감소는 IME의 경우 0.2% 이하로 나타나 매우 작기 때문에, 파이프라인 스케줄로 얻을 수 있는 동작 속도의 이득이 더 크기 때문에 널리 이용 되어 왔다 [40-43].

HEVC에서는 새롭게 MME가 추가되었기 때문에 MME를 기존 3-stage 파이프라인 스케줄에 추가 하기 위해서는 어느 파이프라인 스테이지가 적합한지에 대한 고려가 필요하다. MME의 경우 ME와 다르게 이웃 PU의 움직임 정보에 완전히 의존하고 있기 때문에, 이웃 PU에 대한 최종 모드 결정이 종료되지 않으면 MME를 수행할 수 없다. 이러한

강력한 의존성으로 인해 MME를 가장 마지막 파이프라인 스테이지인 MD 스테이지에서 함께 수행하는 것이 가장 바람직하다. 그림 6.1에 4개의 CU로 이루어진 영상에 대한 3-stage 파이프라인 스케줄의 세가지 예를 나타내었다. 상단은 파이프라인 스케줄을 나타내고 각 직사각형 내의 숫자는 아래에 나타낸 CTU의 인덱스를 나타낸다. 또한 밝은 회색 및 어두운 회색 정사각형은 2Nx2N PU의 merge 후보가 되는 이웃 PU의 위치를 나타낸다. 이 때, 밝은 회색은 이웃 PU의 움직임 정보가 사용 가능한 경우를 나타내며, 어두운 회색은 이웃 PU의 움직임 정보가 사용 가능하지 않은 경우를 나타낸다. 그림 6.1 (a)는 IME와 MME가 같은 파이프라인 스테이지에서 수행되는 경우를 나타내고, 그림 6.1 (b)는 FME와 MME가 같은 파이프라인 스테이지에서 수행되는 경우를 나타내며, 그림 6.2 (c)는 MD와 MME가 같은 파이프라인 스테이지에서 수행되는 경우를 나타낸다. MME는 이웃 PU의 정보를 반드시 필요로 하기 때문에 만약 MD와 MME가 서로 다른 파이프라인 스테이지에서 수행되면 파이프라인 스테이지의 거리에 따라서 사용가능 하지 않은 이웃 PU가 발생한다. 이는 이웃 PU에 대한 MD가 아직 종료되지 않았기 때문에 이웃 PU로부터 움직임 정보를 가져 올 수 없기 때문이다. 그러므로 그림 6.1 (a)는 좌측 및 상단 PU의 움직임 정보가 사용 불가능하고, 그림 6.1 (b)는 좌측 PU의 움직이 정보가 사용 불가능한 것을 볼 수 있다. 반면에 그림 6.1 (c)는 모든 이웃 PU의 움직임 정보가 사용 가능하므로 MME는 MD와 동일한 파이프라인 스테이지에서 동작하는 것이 바람직하다.

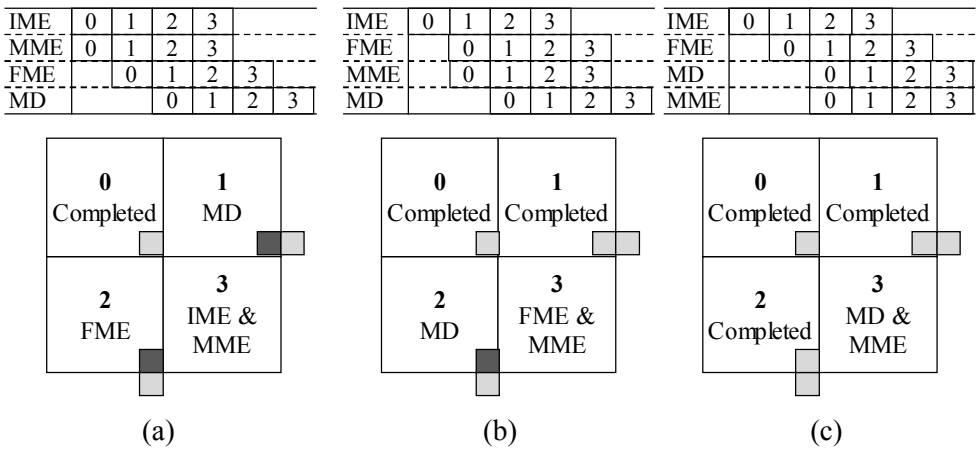


그림 6.1 4 개의 CTU로 이루어진 영상에 대한 3-stage 파이프라인 스케줄의 세 가지 예: (a) IME와 MME가 같은 파이프라인 스테이지, (b) FME와 MME가 같은 파이프라인 스테이지, (c) MD와 MME가 같은 파이프라인 스테이지

6.2 Two-way Encoding Order

6.2.1 Top-down 인코딩 순서와 Bottom-up 인코딩 순서

이 절에서는 3장에서 제안된 bottom-up IME을 HEVC 전체 인코딩 관점에서 살펴본다. 만일 HEVC의 전체 인코딩 과정을 bottom-up 순서를 따르도록 할 경우 현재, top-down 순서를 따르는 다양한 고속 알고리즘을 사용할 수 없게 된다. 결과적으로 계산 복잡도 감소 성능에 상당한 페널티를 받게 된다. Early CU setting (ECU)는 HM 참조 소프트웨어에 탑재되어 있는 고속 알고리즘으로, top-down 순서를 기반으로 하고 있다

[69]. ECU는 상위 CU가 skip 모드로 결정되는 경우 해당 CU 보다 작은 크기에 대한 모든 예측을 수행하지 않도록 한다. ECU는 RD 트레이드 오프 측면에서 매우 효율적인 알고리즘으로 알려져 있다. 만약 특정 CU가 ECU에 의해 인코딩 과정에서 제외되는 경우, IME 뿐 만 아니라, FME와 MME까지 모두 생략되기 때문에, ECU로 인한 계산 복잡도 감소 효과는 IME에만 그치지 않고 인코딩 과정 전체에 적용 된다. 그러므로 오직 IME의 고속화 만을 위해서 bottom-up 순서를 채택하는 것은 다른 top-down 기반 고속 알고리즘들로 인한 계산 복잡도 감소를 사용하지 못하게 되는 큰 문제를 가지게 된다.

6.2.2 기존 고속 알고리즘과 호환되는 Two-way Encoding Order

제안된 bottom-up IME를 다른 top-down 기반의 고속 알고리즘들과 함께 사용 할 수 있도록 하기 위해, two-way 인코딩 순서를 제안한다. Two-way 인코딩 순서에서는 IME만을 bottom-up 순서로 수행하고, 나머지의 다른 연산에 대해서는 top-down 순서로 수행하도록 하는 것이다. 이후로는 이러한 인코딩 순서를 BU-IME/TD-Other로 칭한다. 그림6.2 에 제안하는 BU-IME/TD-Other의 인코딩 순서에 따른 CTU의 인코딩 과정을 pseudo code로 나타내었다. Algorithm 2에 보인 것과 같이 bottom-up 순서로 IME를 수행하여 CTU 내의 모든 PU에 대한 IMV를 얻는다. 다른 FME 및 MME 등의 과정은 Algorithm 3에 나타낸 것과 같이 top-down 순서로 수행되면서 다른 기존의 고속 알고리즘들을 사용할 수 있는 기회를 가질 수 있다. 예를 들어 IME를 수행한 다음 64x64 CU가 ECU의 종료 조건을 만족하면, 이후의 인코딩 과정인 4개의

32x32 CU, 16개의 16x16 CU, 64개의 8x8 CU을 모두 생략 할 수 있다. 그러므로 제안된 BU-IME/RD-Other의 two-way 인코딩 순서는 기존의 고속 알고리즘과 제안한 bottom-up IME를 함께 적용 할 수 있도록 한다. 제안된 two-way 인코딩 순서의 유일한 문제점은 각 PU의 최종 모드 결정이 없이 CTU 내에서 IME를 한번에 수행하므로, 이웃 PU에 대한 정확한 움직임 정보를 알 수 없는 점이다. 이웃 PU의 정확한 움직임 정보를 알 수 없는 경우 AMVP를 정확하게 유도할 수 없게 되고, 이 경우, RD cost 기반의 모드 결정에서 bit 항을 계산할 수 없게 된다. 이러한 문제를 완화하기 위해서 Algorithm 2에 나타낸 것과 같이 pseudo 최적 모드 결정을 IME cost로 수행하도록 하였다. 결과적으로 비록 오차는 있을 수 있지만 AMVP를 유도하여 IME에서 사용할 수 있도록 함으로써 잘못된 최적 모드 결정으로 인한 화질 저하를 최소화 할 수 있다. 시뮬레이션 결과 two-way 인코딩 순서를 사용함으로써 발생하는 압축 효율 저하는 BD-BR 약 0.2%의 증가를 나타내었다. 유도된 pseudo AMVP는 오직 IME의 최적 모드 결정을 위해서만 사용되며, 실제 비트스트림 생성 과정에서는 정확한 AMVP를 유도하여 사용하도록 함으로써 HEVC의 표준을 준수하도록 하였다.

Algorithm 1 Two-way encode CTU

Input: Current CTU, *CTU*
Output: Best CTU coding mode, M_B
1: *Start Depth* ← 0
2: *IMVs for all PUs in CTU* ← Bottom-up IME CU (*Start Depth*)
3: M_B ← Top-down encode CU (*Start Depth*, *IMVs for all PUs in CTU*)
4: **return** M_B

Algorithm 2 Bottom-up IME CU

Input: Current depth, *depth*
Output: IMVs for all PUs in current CU, *IMVs;* Pseudo Best CU coding mode with IME cost for current CU, M_S
1: **if** *depth*<3 then
2: **for** SplitCU=0 **to** 3 **do**
3: *IMVs*, M_S ← Bottom-up IME CU(*depth*+1)
4: **end for**
5: **end if**
6: Perform IME with BMVP for various PU types and store results to *IMVs*
7: Best PU type decision with IME cost
8: **if** Current depth<3 **then**
9: Best depth decision with IME cost, Update M_S
10: **end if**
11: **return** *IMVs*, M_S

Algorithm 3 Top-down encode CU

Input: Current depth, *depth*; IMVs for all PUs in current CU, *IMVs*
Output: Best CU coding mode for current CU, M_B
1: Perform FME, MERGE and residual coding for various PU types with *IMVs*
2: Perform intra prediction and residual coding
3: Best PU type decision
4: **if** *depth*<3 **then**
5: **for** SplitCU=0 **to** 3 **do**
6: M_B ← Top-down encode CU(*depth*+1, *IMVs*)
7: **end for**
8: Best depth decision, Update M_B
9: **end if**
10: **return** M_B

그림 6.2 Two-way 인코딩 순서에 따른 CTU의 인코딩 알고리즘

6.2.3 기존 고속 알고리즘과 결합 및 비교 실험 결과

이 절에서는 제안된 bottom-up IME의 BU-IME/TD-Others의 순서인 two-way 인코딩 순서를 기존의 고속 알고리즘과 결합하였다. 그리고 압축 효율의 변화와 IME 계산 복잡도를 넓은 범위의 QP에 대하여 실험을 통해 구하였다. 비교 실험을 위해, HM 13.0 참조 소프트웨어에 적용된 세 가지 고속 알고리즘을 사용하였다. ECU와 early skip detection (ESD)와 coded block flag fast mode (CFM)이 그것이다 [69-71]. ESD는 단일 CU에 대해 2Nx2N PU의 최적 모드가 skip 모드이거나 또는 MVD가 0이고 coded block flag가 false값인 경우 현재 CU에 대한 다른 PU 타입에 대한 예측 과정을 생략하는 고속 알고리즘이다. CFM은 단일 CU에 대해 다양한 PU 타입의 예측을 수행하면서 도중에 coded block flag가 0인 PU 타입이 나타나면 해당 CU의 예측을 종료하는 고속 알고리즘이다. 유의할 점은 ECU, CFM, ESD는 종료 조건이 만족되면 모든 예측, 즉, IME뿐만 아니라 다른 연산 까지도 모두 생략하는 고속 알고리즘이며, 제안된 bottom-up IME는 오직 IME의 연산만 줄일 수 있는 고속 IME알고리즘이다. 그러므로 제안된 bottom-up IME와 이 알고리즘들을 결합 할 때 two-way 인코딩 순서를 사용하여 IME의 계산 복잡도는 제안된 알고리즘을 사용하여 줄이도록 하였고, 나머지 연산의 계산 복잡도는 기존의 고속 알고리즘의 효과를 가질 수 있도록 하였다.

표 6.1에 제안된 bottom- up IME 알고리즘이 ECU, CFM, ESD와 결합된 실험 결과와 함께 ECU, CFM, ESD만이 사용된 실험 결과를 나타내었다. 제안된 two-way 인코딩 순서의 bottom-up IME는 1 픽셀 크기의 탐색 범위를 사용하였다. 첫 번째와 두 번째 열은 다양한 영상을 나타내며 다른 열들은 압축 효율과 계산 복잡도의 변화를 나타낸다. LDP 조건의

낮은 QP 대역인 QP 7, 12, 17, 22의 ECU, CFM, ESD가 적용된 결과가 세 번째와 네 번째 열에 나타나 있다. 이 때 영상 평균 IME의 계산 복잡도는 9.21% 감소하였다. 반면에 제안된 bottom-up IME로 IME의 계산 복잡도를 줄인 경우의 실험 결과가 다섯 번 째와 여섯 번 째 열에 나타나 있다. 이 경우에는 IME의 계산 복잡도가 78.13% 감소한 것을 볼 수 있다. 또한 공통 실험 조건의 QP인 QP 22, 27, 32, 37대역에 대한 ECU, CFM, ESD만 적용한 실험 결과를 일곱 번 째 열과 여덟 번째 열에 나타내었다. 이 때 BD-BR의 증가는 0.91%이고 IME의 계산 복잡도 감소는 35.81%이다. 반면 아홉 번째 열과 열 번째 열에는 제안된 방법을 함께 적용한 실험 결과를 나타내었다.이 때 BD-BR의 증가는 1.25%이며 IME의 계산 복잡도 감소는 80.82%이다. RA 테스트 조건의 실험 결과에서도 제안된 방법을 함께 적용한 경우 ECU, CFM, ESD만 적용한 것 보다 더 많은 IME계산 복잡도 감소 성능을 보였다.

제안된 bottom-up IME 방법의 우수한 점은 계산 복잡도 성능이 영상 특성에 영향을 적게 받는다는 점이다. 기존 고속 알고리즘은 영상의 특성에 따라서 IME의 계산 복잡도 감소 성능이 크게 영향을 받는 다는 점에 주목하라. 예를 들어 ECU, CFM, ESD만 적용된 경우 공통 QP조건에서 IME 계산 복잡도 감소는 LDP에 대해 66.23%에서 17.37%까지 매우 큰 폭으로 변화한다. 반면에 제안된 방법을 적용한 경우 계산 복잡도 감소는 모든 영상에서 LDP에 대해 88.05%에서 71.97%까지, RA에 대해 80.54%에서 62.43%까지로 적은 폭의 성능 변화를 보인다. 게다가 기존 고속 알고리즘은 복잡한 움직임을 포함하는 영상에 대해서 잘 작동하지 않는다. 움직임이 복잡하고 빠른 영상들은 높은 IME 계산 복잡도를 야기하는 데에도 불구하고 기존 고속 알고리즘은 계산 복잡도를 감소시키지 못하는 것이다. 즉, 고속

알고리즘으로 계산 복잡도를 감소시킬 필요성이 매우 높지만, 이 때에 오히려 고속 알고리즘이 제대로 작동하지 못한다는 의미이다. 예를 들어 LDP의 공통 QP조건에서 움직임이 복잡한 PeopleOnStreet, BasketballDrive, RaceHorses, BlowingBubbles 영상에 대해 기존 고속 알고리즘인 ECU, CFM, ESD는 단지 16.57% 에서 27.02% 의 계산 복잡도 감소 성능을 보인다. 반면에 BQTerrace와 Johnny와 같이 단순한 움직임을 갖는 영상에대해서는 62.93%, 66.24%의 계산 복잡도 성능을 가진다. 이러한 경향성은 RA 테스트 조건에서도 나타난다. 그러나, 제안된 알고리즘은 움직임이 복잡한 영상이나 단순한 영상에서도 우수한 계산 복잡도 감소 성능을 보인다. 예를 들어 LDP 조건에서 PeopleOnStreet와 BasketballDrill과 RaceHorses 영상에 대해서 IME 계산 복잡도 감소 성능은 80% 이상을 보인다. 이러한 계산 복잡도 감소 성능의 안정성은 낮은 QP 대역에서도 나타난다. 예를 들어 ECU, CFM, ESD가 적용된 경우 낮은 QP 대역에서 IME 계산 복잡도 감소 성능은 영상 평균 9.21%에 불과하지만 제안된 방법이 적용된 경우 78.13%로 여전히 높은 계산 복잡도 감소 성능을 보인다. 이는 기존의 고속 알고리즘들은 skip 모드나 residual이 발생하는가에 대한 정보에 의존하기 때문에 residual이 많이 발생되는 낮은 QP 대역에서는 제대로 작동하지 못하기 때문이다.

그러므로 제안된 bottom-up IME는 입력 영상 특성과 QP 설정값에 덜 영향을 받기 때문에 매우 실용적이다.

표 6.1 Two-way 인코딩 순서가 적용된 bottom-up IME와 HM의 ECU, CFM, ESD를 결합한 실험 결과와 ECU, CFM, ESD만 적용하였을 때의 압축 효율 및 IME 계산 복잡도 비교

Video Sequence		Low-delay P							
		QP 7,12,17,22				QP 22,27,32,37			
		ECU+CFM+ESD		Proposed (SR=1)		ECU+CFM+ESD		Proposed (SR=1)	
		BR(%)	ΔC(%)	BR(%)	ΔC (%)	BR(%)	ΔC(%)	BR(%)	ΔC (%)
ClassA	Traffic	0.35	-10.91	0.61	-69.84	0.97	-48.84	1.15	-71.97
	PeopleOnStreet	0.06	-1.77	0.14	-82.59	1.06	-17.37	2.01	-85.38
ClassB	BasketballDrive	0.04	-4.33	0.07	-86.30	0.74	-30.10	1.40	-88.05
	BQTerrace	0.09	-13.30	0.14	-77.06	0.85	-62.93	0.97	-78.19
	Cactus	0.15	-3.69	0.17	-77.97	1.62	-28.71	1.83	-83.07
	Kimono	0.06	-4.89	0.06	-84.11	0.55	-28.24	0.81	-86.52
	ParkScene	0.14	-8.04	0.19	-78.74	1.19	-42.93	1.41	-78.98
ClassC	BQMall	0.00	-12.14	0.17	-80.37	0.98	-28.09	1.37	-83.61
	BasketballDrill	0.10	-10.36	0.13	-80.89	0.54	-25.71	0.98	-84.21
	PartyScene	0.16	-5.32	0.17	-75.81	1.01	-65.30	1.23	-78.35
	RaceHorses	0.06	-4.74	0.13	-84.87	0.80	-18.42	1.58	-87.36
ClassD	BasketballPass	0.25	-12.87	0.37	-80.65	1.21	-30.66	1.66	-83.83
	BlowingBubbles	0.13	-3.67	0.03	-71.52	1.43	-27.02	1.39	-75.50
	RaceHorses	0.08	-1.66	0.15	-81.13	1.15	-16.57	1.92	-84.71
ClassE	FourPeople	0.58	-18.13	0.60	-69.21	0.11	-58.15	0.28	-73.72
	Johnny	0.57	-22.00	0.42	-73.19	1.07	-66.24	0.94	-74.02
	KristenAndSara	0.70	-18.70	0.74	-74.03	0.24	-59.98	0.28	-76.49
	Average	0.21	-9.21	0.25	-78.13	0.91	-35.81	1.25	-80.82
Video Sequence		Random Access							
		QP 7,12,17,22				QP 22,27,32,37			
		ECU+CFM+ESD		Proposed (SR=1)		ECU+CFM+ESD		Proposed (SR=1)	
		BR(%)	ΔC (%)	BR(%)	ΔC (%)	BR(%)	ΔC (%)	BR(%)	ΔC (%)
ClassA	Traffic	0.56	-23.10	0.52	-46.35	1.57	-65.40	1.65	-68.63
	PeopleOnStreet	0.10	-4.10	0.25	-66.29	1.77	-25.40	3.14	-72.73
ClassB	BasketballDrive	0.15	-8.85	0.11	-76.60	1.13	-38.08	2.09	-80.54
	BQTerrace	0.18	-20.62	0.20	-58.19	1.01	-73.06	1.23	-74.67
	Cactus	0.23	-10.51	0.27	-59.60	1.96	-45.41	2.49	-75.39
	Kimono	0.20	-8.39	0.14	-70.33	1.18	-37.07	1.49	-74.56
	ParkScene	0.31	-16.59	0.31	-59.41	1.48	-55.83	1.69	-68.54
ClassC	BQMall	0.30	-11.98	0.41	-59.65	2.31	-39.21	2.72	-72.02
	BasketballDrill	0.36	-12.79	0.39	-63.76	1.16	-36.90	2.05	-74.23
	PartyScene	0.08	-9.86	0.14	-56.00	1.22	-35.89	1.61	-68.24
	RaceHorses	0.07	-4.01	0.14	-72.48	2.18	-24.58	3.14	-76.74
ClassD	BasketballPass	0.55	-23.08	0.62	-65.06	2.04	-66.51	2.46	-72.83
	BlowingBubbles	0.20	-10.25	0.28	-47.75	1.67	-38.13	1.83	-62.43
	RaceHorses	0.14	-4.07	0.23	-64.67	2.58	-22.13	3.84	-70.46
ClassE	FourPeople	0.46	-33.26	0.47	-51.12	0.32	-77.94	0.47	-78.08
	Johnny	0.53	-35.45	0.47	-54.75	0.87	-80.97	0.92	-77.18
	KristenAndSara	0.85	-34.89	0.83	-55.61	0.58	-77.28	0.68	-76.16
	Average	0.31	-15.10	0.34	-57.09	1.47	-45.32	1.97	-69.08

제 7 장 Next Generation Video Coding으로의 확장

7.1 Bottom-up Motion Vector Prediction의 확장

3.2절에서 제안한 bottom-up MVP는 IME를 위해서만 사용되는 인코더의 고속화를 위한 MV 예측 방법이다. 제안된 MV 예측 방법은 기존의 공간적, 시간적으로 인접한 PU로부터 MV를 예측하는 방법 보다 우수한 MV 예측 성능을 보여주었다. 그러나, 이 예측 방법을 차세대 영상 코딩 표준에서 적용할 수 있다면, MV 코딩을 위한 비트량 사용을 더욱 감소 시킬 수 있을 것이다. 그러나, 제안된 MV 예측 방법은 최하위 depth 의 움직임 정보를 사용하여 상위 depth의 MV를 예측 하는 방법으로써, 하나의 depth만을 선택하여 영상을 인코딩 하여 디코더로 전송하는 영상 압축 방식에서는, 디코더 단에서 하위 depth의 MV를 알 수 없으므로 사용되기 어렵다. 그러므로 만일 최하위 depth의 움직임 벡터를 디코더로 전송하여 상위 depth의 움직임 벡터를 예측하고자 한다면, 오히려 비트의 사용량을 증가시킬 위험이 있다. 그러므로 bottom-up MV 예측 방법을 표준에 직접 적용하는 것은 어렵다. 그러나, 차세대 압축 표준을 준비하는 moving picture experts group (MPEG)에서는 제안된 bottom-up MV 예측의 다수의 하위 PU로부터 MV를 예측하는 방법과 유사한 방법의 advanced temporal motion vector prediction (ATMVP)를 차세대 영상 압축 표준에 채택하려는 움직임을 보이고 있다. ATMVP는 현재 PU를 위해 co-located picture로부터 다수의 subCU를

사용하여 움직임 벡터를 예측하는 방법이다. 그림 7.1에 ATMVP의 유도 방법에 대하여 나타내었다. ATMVP는 co-located picture의 MV field를 사용하는데, MV field는 이미 디코딩 된 co-located picture의 MV 정보를 말한다. HEVC의 경우 MV 저장 공간의 절약을 위해서 MV field를 16x16 픽셀 크기 단위로 생성하였는데, 차세대 압축 표준에서는 MV field를 최소 블록 단위인 4x4로 구성하여 세밀한 MV예측이 가능하도록 하였다. Co-located picture 내에서 co-located PU를 구하는 방법을 개선하였다. HEVC에서는 co-located PU의 위치를 그림 2.3의 C와 H 위치로 한정하였지만, 서로 다른 시간 축 상의 picture의 경우 동일한 픽셀 영역이 다른 위치로 이동하였을 수 있으므로, 이를 반영하기 위해 첫 번째 merge 후보의 MV값 만큼 이동한 위치를 co-located PU로 정하였다. 다음으로 co-located PU위치에서 현재 예측 하고자 하는 PU의 크기 내의 subCU들을 기준으로 세분화된 MV field를 이용하여 다수의 MV를 참조한다. 이렇게 참조된 MV는 기존의 temporal MV 예측 방법에 의해 스케일링 되어 MVP로 사용된다. ATMVP에서는 co-located PU 위치에서 세분화된 MV field를 참조함으로써, bottom-up MV 예측이 하위 CU로부터 여러 개의 MVP를 유도하는 것과 동일한 아이디어를 활용하고 있다. 만약 ATMVP가 참조하는 MVfield영역이 가장 작은 PU로 코딩 되었고 현재 MV를 코딩하고자 하는 PU의 크기가 16x16 PU라면, 제안된 MV예측 방법이 8x8 CU로부터 상위 depth인 16x16 PU의 MV를 예측하는 것과 동일한 동작을 하게 된다.

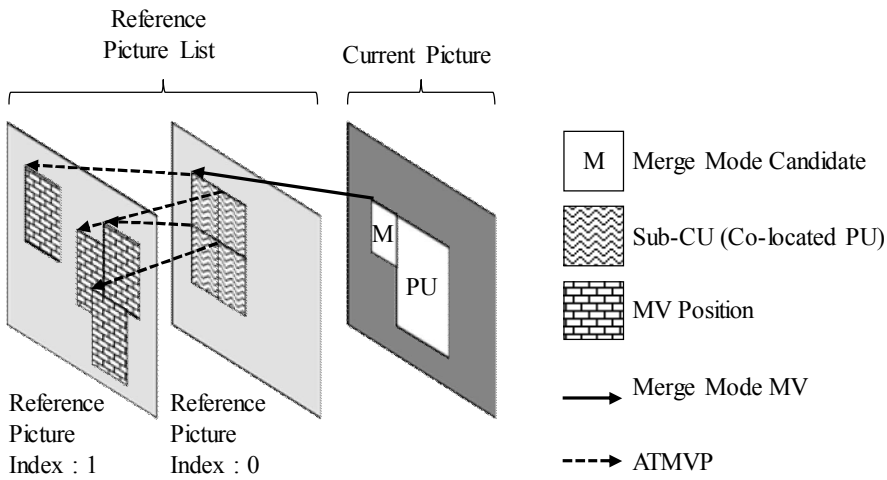


그림 7.1 차세대 압축 표준의 ATMVP를 유도하는 과정

본 논문에서는 이 BMVP의 가능성을 확인 해 보기 위해서, ATMVP의 방법과 동일하게 MV 예측을 수행하여 보았다. 그리고 예측된 MVP를 기존 HEVC의 temporal MVP 유도 과정에 추가하여 최대 3개의 AMVP를 사용할 수 있도록 하여 비트스트림을 생성하였다. 표 7.1에 ATMVP를 HEVC에서 활용하였을 때의 압축 효율 변화를 나타내었다. 실험 결과 모든 컬러 성분과 모든 테스트 조건에서 적은 폭이지만 BD-BR의 개선 효과를 확인 할 수 있었다. 차세대 압축 표준의 제정과정에서 더욱 자세한 연구가 이루어진다면 본 논문의 아이디어를 기반으로 한 발전된 MV 예측 방법으로 더 나은 압축 효율 개선 성능을 보일 수 있을 것이다.

표 7.1 차세대 압축 표준의 ATMVP를 HEVC에 적용하였을 때의 압축 효율 변화

Video Sequence	Low-delay P			Random Access		
	Y	U	V	Y	U	V
Class A	-0.21	-0.30	-0.20	-0.33	-0.23	-0.41
Class B	0.03	0.17	0.07	-0.07	-0.03	0.11
Class C	-0.20	-0.23	-0.44	-0.14	0.14	-0.04
Class D	-0.17	-0.67	-0.15	-0.05	-0.05	0.00
Class E	0.15	0.06	-0.40	-0.05	-0.05	0.00
Average	-0.07	-0.18	-0.20	-0.07	0.00	-0.04

7.2 Bottom-up Integer Motion Estimation의 확장

2017년, 현재, MPEG에서 차세대 비디오 코딩 표준의 제정 작업이 진행 중이다 [JEM]. 차세대 비디오 코딩 표준은 더 높은 해상도의 영상에 대한 인코딩을 효율적으로 수행하기 위해 CTU의 크기를 128x128로 확장하여 테스트를 진행하고 있다. 또한 기존의 쿼드-트리 블록 계층 구조를 확장하여 더 높은 자유도를 갖는 바이너리-트리 블록 계층 구조를 도입하였다. 그러므로 선택 가능한 계층의 수가 증가함에 따라 IME의 복잡도 또한 증가할 것이다. 또한 차세대 압축 표준이 지원할 수 있는 더욱 높은 해상도의 영상을 압축하기 위해서는 더욱 넓은 탐색 범위를 탐색해야 할 것이다. 그러므로 BMA와 MC 기반의 inter prediction을 사용하는 영상 압축 표준에서는, IME 계산 복잡도는 보다 높은 압축 성능을 얻기 위해 계속해서 증가하게 될 것이다.

지속적으로 증가할 IME의 계산 복잡도에 대응하기 위해 새로운 고속 IME 알고리즘이 계속해서 연구되겠지만, 본 논문에서 제안한 bottom-up IME는 블록 계층의 수가 많아짐에 따라 증가하는 IME의 계산 복잡도

증가 문제를 해결할 수 있는 방법이 될 수 있다. 왜냐하면, 제안된 bottom-up IME는 계층의 수가 더욱 증가하더라도 최하위 depth가 아닌 경우, 매우 적은 양의 추가 계산 만으로 IME의 수행 가능하기 때문이다. 그러므로 계층 수가 더욱 증가된 차세대 영상 압축 표준에 제안된 bottom-up IME를 적용한다면, IME의 계산 복잡도 증가에 대한 우려는 본 논문을 통해 종식될 수 있을 것이다.

제 8 장 결 론

본 연구에서는 HEVC를 위한 고속 IME 알고리즘과 함께 실시간 인코딩을 위한 하드웨어 구조를 제안하였다. 기존의 고속 IME 알고리즘들은 계산 복잡도를 줄이기 위해서 압축 효율을 희생하는 접근 방식을 사용해 왔기 때문에, HEVC가 가진 본래의 영상 압축 성능을 달성 할 수 없었다. 그러나 본 연구에서 제안한 bottom-up MV 예측 방법은 기존의 공간적, 시간적으로 인접한 PU로부터 MV를 예측하는 방법이 아닌, HEVC의 계층적으로 인접한 PU로부터 MV를 예측하는 방법을 제안하여 MV 예측의 정확도를 큰 폭으로 향상시켰다. 결과적으로 압축 효율의 변화 없이 IME의 계산 복잡도를 67% 감소시킬 수 있었다. 제안된 bottom-up IME 알고리즘을 적용하여 실시간 동작이 가능한 하드웨어 기반의 IME를 제안하였다. 기존의 하드웨어 기반 IME는 고속 IME 알고리즘이 갖는 단계별 의존성으로 인한 idle cycle의 발생과 참조 데이터 접근 문제로 인해, 고속 IME 알고리즘을 사용하지 않거나 또는 하드웨어에 맞게 고속 IME 알고리즘을 수정하였기 때문에 계산 복잡도 감소 성능이 우수하지 않았다. 그러나 본 연구에서는 고속 IME 알고리즘인 TZS 알고리즘을 채택하여 TZS 알고리즘의 계산 복잡도 성능을 훼손하지 않는 하드웨어 기반의 IME를 제안하였다. 고속 IME 알고리즘을 하드웨어에서 사용하기 위해서 다음 세 가지 사항을 제안하고 하드웨어에 적용하였다. 첫 째로, 고속 IME 알고리즘의 고질적 문제인 idle cycle 발생 문제를 서로 다른 참조 픽쳐와 서로 다른 depth에 대한 IME를 컨텍스트 스위칭을 통해 해결하였다. 둘 째로, 참조 데이터로의 빠르고 자유로운 접근을 위해 참조 데이터의 locality 이용한

multi bank SRAM 구조를 제안하였다. 셋 째로, 지나치게 자유로운 참조 데이터 접근이 발생시키는 대량의 스위칭 mux의 사용을 피하기 위해 탐색 중심을 기준으로 하는 제한된 자유도의 참조 데이터 접근을 제안하였다. 결과 제안된 IME 하드웨어는 HEVC의 모든 블록 크기를 지원하면서, 참조 픽처 4장를 사용하여, 4k UHD 영상을 60fps의 속도로 처리할 수 있으며 이 때 압축 효율의 손실은 0.40%로 거의 나타나지 않는다. 이 때 사용되는 하드웨어 리소스는 1.27M gates이다.

HEVC에 새로이 채택된 merge mode estimation은 압축 효율 개선 효과가 뛰어난 새로운 기술이지만, 매 PU 마다 계산 복잡도의 변동 폭이 커서 하드웨어로 구현되는 경우 하드웨어 리소스의 낭비가 많다. 그러므로 본 연구에서는 효율적인 하드웨어 기반 MME 방법과 하드웨어 구조를 함께 제안하였다. 기존 MME 방식은 이웃 PU에 의해 보간 필터 적용 여부가 결정되기 때문에, 보간 필터의 사용률은 50% 이하를 나타낸다. 그럼에도 불구하고 하드웨어는 보간 필터를 사용하는 경우에 맞추어 설계되어왔기 때문에 하드웨어 리소스의 사용 효율이 낮았다. 본 연구에서는 가장 하드웨어 리소스를 많이 사용하는 세로 방향 보간 필터를 절반 크기로 줄인 두 개의 데이터 패스를 갖는 MME 하드웨어 구조를 제안하였고, 높은 하드웨어 사용률을 유지하면서 압축 효율 손실을 최소화 하는 merge 후보 할당 알고리즘을 제안하였다. 결과, 기존 하드웨어 기반 MME 보다 24% 적은 하드웨어 리소스를 사용하면서도 7.4% 더 빠른 수행 시간을 갖는 새로운 하드웨어 기반의 MME를 달성하였다. 제안된 하드웨어 기반의 MME는 460.8K gates의 하드웨어 리소스를 사용하고 4k UHD 영상을 30 fps의 속도로 처리할 수 있다.

참고 문헌

[1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T REC, H.264|ISO/IEC 11496-10 AVC), March 2003.

[2] Recommendation ITU-T H.265, MPEG H -- Part 2: High efficiency video coding, ISO/IEC 23008-2, 2013.

[3] JCT-VC, (2014, Jan.). High Efficiency Video Coding Reference Software. High Efficiency Video Coding Test Model 13.0 (HM13.0). [Online]. Available: http://hevc.hhi.fraunhofer.de/

[4] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," presented at the 13th VCEG-M33 Meeting, Austin, Texas, USA, Apr. 2-4, 2001.

[5] J. Chalidabhongse and C.-C. J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," IEEE Trans. Circuits and Systems for Video Technology, vol. 7, no. 3, pp. 477–488, Jun. 1997.

[6] S. D. Kim and J. B. Ra, "An efficient motion vector coding scheme based on minimum bitrate prediction," IEEE Trans. Image Processing, vol. 8, no. 8, pp. 1117–1120, Aug. 1999.

[7] K.R. Namuduri, "Motion estimation using spatio-temporal contextual information," IEEE Trans. Image Processing, vol. 14, no. 8, pp. 1111–1115, Aug. 2004.

[8] W.-D. Chien, K.-Y. Kiao, and J.-F. Yang, "Enhanced AMVP mechanism based adaptive motion search range decision algorithm for fast HEVC coding," in Proc. ICIP., Paris, France, 2014, pp. 3696–3699.

[9] S. Kappagantula and K. Rao, "Motion Compensated Interframe Image Prediction," IEEE Trans. Communications, vol. 33, no. 9, pp. 1011–1015, Sep. 1985.

[10] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for

block motion estimation," IEEE Trans. Circuits and Systems for Video Technology, vol. 4, no. 4, pp. 438–442, Aug. 1994.

[11] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," IEEE Trans. Image Processing, vol. 9, no. 2, pp. 287–290, Feb. 2000.

[12] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," IEEE Trans. Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 313–317, Jun. 1996.

[13] J.-Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation," IEEE Trans. Circuits and Systems for Video Technology, vol. 8, no. 4, pp. 369–377, Aug. 1998.

[14] C.-H. Cheung and L.-M. Po, "A Novel Cross-Diamond Search Algorithm for Fast Block Motion Estimation," IEEE Trans. Circuits and Systems for Video Technology, vol. 12, no. 12, pp. 1168–1177, Dec. 2002.

[15] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Hexagon-based search pattern for fast block motion estimation," IEEE Trans. Circuits and Systems for Video Technology, vol. 12, no. 5, pp. 349–355, May 2002.

[16] Z. Li and Q. Yang, "A Fast Adaptive Motion Estimation Algorithm," in Proc. ICCSEE., Hangzhou, China, 2012, pp. 656-660.

[17] C.-S. Yu and S.-C. Tai, "Adaptive Double-Layered Initial Search Pattern for Fast Motion Estimation," IEEE Trans. Multimedia, vol. 8, no. 6, pp. 1109–1116, Dec. 2006.

[18] J. Wei, H. Fan, and X. Wang, "Adaptive Multi-pattern Search Algorithm for Motion Estimation," in Proc. ICIEA., Xi'an, China, 2009, pp. 3728-3731.

[19] X. Li, R. Wang, X. Cui, and W. Wang, "Context-adaptive fast motion estimation of HEVC," in Proc. ISCAS., Lisbon, Portugal, 2015, pp. 2784-2787.

[20] C.-M. Kuo, Y.-H. Kuan, C.-H. Hsieh, and Y.-H. Lee, "A Novel Prediction-Based Directional Asymmetric Search Algorithm for Fast Block-Matching Motion Estimation," IEEE Trans. Circuits and Systems for Video Technology,

vol. 19, no. 6, pp. 893–899, Mar. 2009.

[21] N. Purnachand, L. N. Alves, and A. Navarro, “Improvements to TZ search motion estimation algorithm for multiview video coding,” in Proc. IWSSIP., Vienna, Austria, 2012, pp. 388-391.

[22] H. Kibeya, F. Belghith, H. Loukil, M. A. B. Ayed, and N. Masmoudi, “TZSearch Pattern Search Improvement for HEVC Motion Estimation Modules,” in Proc. ATSIP., Sousse, Tunisia, 2014, pp. 95-99.

[24] X. Li, R. Wang, W. Wang, Z. Wang, and S. Dong, “Fast Motion Estimation Methods for HEVC,” in Proc. International Symposium on BMSB., Beijing, China, 2014, pp. 1-4.

[25] G.-L. Li and C.-H. Yen, “Clock cycle oriented data bandwidth aware merge mode motion vector selection algorithm for HEVC,” in Proc. SII., Fukuoka, Japan, 2012, pp. 901-905.

[26] M. Kim, H.-J. Lee, and N. Ling, “Fast merge mode decision for diamond search in High Efficiency Video Coding,” in Proc. VCIP., Kuching, Malaysia, 2013, pp. 1-6.

[27] C.-Y. Chen, S.-Y. Chien, T.-C. Chen, T.-C. Wang, and L.-G. Chen, “Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC,” IEEE Trans. on Circuits and Systems, vol. 53, no. 3, pp.578-593, Mar 2006.

[28] C. E. Rhee, J.-S. Jung, and H.-J. Lee, “A Real-time H.264/AVC Encoder with Complexity-Aware Time Allocation,” IEEE Trans. Circuits and Systems for Video Technology, vol. 20, no. 12, pp. 1848–1862, Dec. 2010.

[29] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, “On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture,” IEEE Trans. Circuits and Systems for Video Technology, vol. 12, no. 1, pp. 61–72, Jan. 2002.

[30] C.-Y. Chen, C.-T. Huang, Y.-H. Chen, and L.-G. Chen, “Level C+ data reuse scheme for motion estimation with corresponding coding orders,” IEEE Trans. Circuits and Systems for Video Technology, vol. 16, no. 4, pp. 553–558, Apr. 2006.

[31] C.-Y. Tsai, C.-H. Chung, Y.-H. Chen, T.-C. Chen, and L.-G. Chen, "Low Power Cache Algorithm and Architecture Design for Fast Motion Estimation in H.264/AVC Encoder System," in Proc. ICASSP., Honolulu, HI, USA, 2007, pp. II-97 - II-100.

[32] P.-K. Tsung, W.-Y. Chen, L.-F. Ding, S.-Y. Chien, and L.-G. Chen, "Cache-based integer motion/disparity estimation for quad-HD H.264/AVC and HD multiview video coding," in Proc. ICASSP., Taipei, Taiwan, 2009, pp. 2013-2016.

[33] J. Byun, Y. Jung and, J. Kim, "Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD," IEEE Electronics Letters, vol. 49, no. 18, pp. 1142–1143, Aug. 2013.

[34] M. E. Sinagil, V. Sze, Minhua Zhou, and A. P. Chandrakasan, "Cost and Coding Efficient Motion Estimation Design Considerations for High Efficiency Video Coding (HEVC) Standard," IEEE Journal. Selected Topics in Signal Processing, vol. 7, no. 6, pp. 1017–1028, Jul. 2013.

[35] S.-Y. Jou, S.-J. Chang, and T.-S. Chang, "Fast Motion Estimation Algorithm and Design for Real Time QFHD High Efficiency Video Coding," IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 9, pp. 1533–1544, Sep. 2015.

[36] G. Pastuszak and M. Trochimiuk, "Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder," J Real-Time Image Proc., vol. 12, no. 2, pp. 517–529, Jul. 2015.

[37] G. Sanchez, M. Porto, L. Agostini, "A hardware friendly motion estimation algorithm for the emergent HEVC standard and its low power hardware design," in Proc. ICIP., Melbourne, Australia, 2013, pp. 1991-1994.

[38] C. Machado Diniz, M. Shafique, S. Bampi, and J. Henkel, "High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for HEVC," in Proc. ICIP., Melbourne, Australia, 2013, pp. 2091-2095.

[39] P. Helle, S. Oudin, B. Bross, D. Marpe, M.O. Bici, K. Ugur, J. Jung, G. Clare, and T. Wiegand, "Block Merging for Quadtree-Based Partitioning in HEVC," IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12,

pp. 1720–1731, Dec. 2012.

[40] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," IEEE Trans. Circuits and Systems for Video Technology, vol. 16, no. 6, pp. 673–688, Jun. 2006.

[41] J.H. Lee and N.S. Lee, "Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC," in Proc. ISCAS., Vancouver, Canada, 2004, pp. III - 741-4.

[42] Y.-K. Lin, C.-C. Lin, T.-Y. Kuo, and T.-S. Chang, "A Hardware-Efficient H.264/AVC Motion-Estimation Design for High-Definition Video," IEEE Trans. Circuits and Systems, vol. 55, no. 6, pp. 1526–1535, Feb. 2008.

[43] L. Zhang and W. Gao, "Reusable Architecture and Complexity-Controllable Algorithm for the Integer/Fractional Motion Estimation of H.264," IEEE Trans. Consumer Electronics, vol. 53, no. 2, pp. 749–756, May 2007.

[44] J. Chen, E. Alshina, G. J. Sullivan, J-. R. Ohm, and J. Boyce, "Algorithm Description of Joint Exploration Test Model 5 (JEM 5)," JVET document, JVET-E1001-v2, Jan. 2017.

[45] R. De Forni and D.S. Taubman, "On the benefits of leaf merging in quad-tree motion models," in Proc. ICIP., Genova, Italy, 2005, pp. II - 858-61.

[46] R. Mathew and D.S. Taubman, "Quad-Tree Motion Modeling with Leaf Merging," IEEE Trans. Circuits and Systems for Video Technology, vol. 20, no. 10, pp. 1331–1345, Sep. 2010.

[47] Rahul Shukla, P.L. Dragotti, M.N. Do, and M. Vetterli, "Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images," IEEE Trans. Image Processing, vol. 14, no. 3, pp. 343–359, Mar. 2005.

[48] D. Woo, C. E. Rhee, and H.-J. Lee, "A cache-aware motion estimation organization for a hardware-based H.264 encoder," IEEE Trans. Consumer Electronics, vol. 60, no. 1, pp. 83–91, Feb. 2014.

[49] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of H.264/AVC," in Proc. ICASSP.,

Montreal, Quebec, Canada, 2004, pp. V - 9-12.

[50] G. Kim, J. Kim, and C.-M. Kyung, “A low cost single-pass fractional motion estimation architecture using bit clipping for H.264 video codec,” in Proc. ICME., Singapore, Singapore, 2010, pp. 661-666.

[51] G. He, D. Zhou, Z. Chen, T. Zhang, and S. Goto, “A 995Mpixels/s 0.2nJ/pixel fractional motion estimation architecture in HEVC for Ultra-HD,” in Proc. A-SSCC., Singapore, Singapore, 2013, pp. 301-304.

[52] X. Lian, W. Zhou, Z. Duan, and R. Li, “An efficient interpolation filter VLSI architecture for HEVC standard,” in Proc. China-SIP., Xi'an, China, 2014, pp. 384-388.

[53] J. Liu, X. Chen, Y. Fan, and X. Zeng, “A full-mode FME VLSI architecture based on 8×8/4×4 adaptive Hadamard Transform for QFHD H.264/AVC encoder,” in Proc. VLSI-SoC., Kowloon, Hong Kong, 2011, pp. 434-439.

[54] C. Yang, S. Goto, and T. Ikenaga, “High performance VLSI architecture of fractional motion estimation in H.264 for HDTV,” in Proc. ISCAS., Kos, Greece, 2006, pp. 2605-2608.

[55] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, “Overview of the High Efficiency Video Coding (HEVC) Standard,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[56] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, “Block Partitioning Structure in the HEVC Standard,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.

[57] S. Kamp and M. Wien, “Decoder-Side Motion Vector Derivation for Block-Based Video Coding,” IEEE Trans. Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1732–1745, Dec. 2012.

[58] B. Liu and A. Zaccarin, “New Fast Algorithms for the Estimation of Block Motion Vectors,” IEEE Trans. Circuits and Systems for Video Technology, vol. 10, no. 3, pp. 417–422, Apr. 2000.

[59] C.-K. Cheung and L.-M. Po, “Normalized Partial Distortion Search Algorithm for Block Motion Estimation,” IEEE Trans. Consumer Electronics,

vol. 58, no. 4, pp. 1375–1383, Nov. 2012.

[60] S. Hwang and M. H. Sunwoo, “Efficient Integer Motion Estimation Algorithm using Sub-sampling,” in Proc. ISOCC., Busan, Korea, 2009, pp. 361-364.

[61] H. Yin, H. Jia, H. Qi, X. Ji, X. Xie, and W. Gao, “A Hardware-Efficient Multi-Resolution Block Matching Algorithm and Its VLSI Architecture for High Definition MPEG-Like Video Encoders,” IEEE Trans. Circuits and Systems for Video Technology, vol. 20, no. 9, pp. 1242–1254, Sep. 2010.

[62] Y.-T. Chang and W.-H. Chung, “A dynamic search range algorithm for H.264/AVC full-search motion estimation,” in Proc. APCCAS., Kuala Lumpur, Malaysia, 2010, pp. 124-127.

[63] K. M. Nam, J.-S. Kim, R.-H. Park, and Y. S. Shim, “A Fast Hierarchical Motion Vector Estimation Algorithm Using Mean Pyramid,” IEEE Trans. Circuits and Systems for Video Technology, vol. 5, no. 4, pp. 344–351, Aug. 1995.

[64] B. C. Song and K.-W. Chun, “Multi-Resolution Block Matching Algorithm and Its VLSI Architecture for Fast Motion Estimation in an MPEG-2 Video Encoder,” IEEE Trans. Circuits and Systems for Video Technology, vol. 14, no. 9, pp. 1119–1137, Sep. 2004.

[65] X. Song, T. Chiang, and Y.-Q. Zhang, “A hierarchical motion estimation algorithm using nonlinear pyramid for MPEG-2,” in Proc. ISCAS., Hong Kong, 1997, vol.2, pp. 1165-1168.

[66] K.-C. Hou, M.-J. Chen, and C.-T. Hsu, “Fast Motion Estimation by Motion Vector Merging Procedure for H. 264,” in Proc. ICME., Amsterdam, Nederland, 2005, pp. 1444-1447.

[67] Y.-K. Tu, J.-F. Yang, Y.-N. Shen, and M.-T. Sun, “Fast variable-size block motion estimation using merging procedure with an adaptive threshold,” in Proc. ICME., Baltimore, Maryland, USA, 2003, pp. II-789-792.

[68] A. Chang, P. H. W. Wong, Y. M. Yeung, and O, C, Au, “Fast integer motion estimation for H.264 video coding standard,” in Proc. ICME., Taipei, Taiwan, 2004, pp. I-289-292.

[69] K. Choi, S.-H. Park, and E. S. Jang, “Coding tree pruning based CU early termination,” JCT-VC document, JCTVC-F092, Jul. 2011.

[70] J. Yang, J. Kim, K. Won, H. Lee, and B. Jeon, “Early skip detection for HEVC,” JCT-VC document, JCTVC-G543, Nov. 2011.

[71] R. H. Gweon, Y. -L. Lee, and J. Lim, “Early Termination of CU Encoding to Reduce HEVC Complexity,” JCT-VC document, JCTVC-F045, Jul. 2011.

[72] J. Zhang, B. Li, and H. Li, “An Efficient Fast Mode Decision Method for Inter Prediction in HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 26, no. 8, pp. 1502–1515, Aug. 2016.

[73] L. Shen, Z. Zhang, and Z. Liu, “Adaptive Inter-Mode Decision for HEVC Jointly Utilizing Inter-Level and Spatiotemporal Correlations,” IEEE Trans. Circuits and Systems for Video Technology, vol. 24, no. 10, pp. 1709–1722, Oct. 2014.

[74] J. Vanne, M. Viitanen, and T. D. Hämäläinen, “Efficient Mode Decision Schemes for HEVC Inter Prediction,” IEEE Trans. Circuits and Systems for Video Technology, vol. 24, no. 9, pp. 1579–1593, Sep. 2014.

[75] W. Zhao, T. Onoye, and T. Song, “Hierarchical Structure-Based Fast Mode Decision for H.265/HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 10, pp. 1651–1664, Oct. 2015.

[76] Z. Pan, S. Kwong, M. Sun, J. Lei, “Early MERGE Mode Decision Based on Motion Estimation and Hierarchical Depth Correlation for HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 20, no. 6, pp. 925–930, Jun. 2010.

[77] S. Ahn, B. Lee, M. Kim, “A Novel Fast CU Encoding Scheme Based on Spatiotemporal Encoding Parameters for HEVC Inter Coding,” IEEE Trans. Circuits and Systems for Video Technology, vol. 25, no. 3, pp. 422–435, Mar. 2015.

[78] J. Zhang, B. Li, and H. Li, “An Efficient Fast Mode Decision Method for Inter Prediction in HEVC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 26, no. 8, pp. 1502–1515, Aug. 2016.

Abstract

Hardware based High Accuracy Integer Motion Estimation and Merge Mode Estimation

Tae Sung Kim
Electrical and Computer Engineering
The Graduate School
Seoul National University

HEVC has twice as much compression efficiency as H.264 / AVC, but using a lot of compression techniques has greatly increased the computational complexity of the encoder side. Much research has been done to reduce the high computational complexity of HEVCs, but most of the studies have only extended the application of computational complexity reduction methods for H.264 / AVC. Therefore, it is not possible to achieve satisfactory computational complexity reduction performance or to achieve a potential compression performance of the HEVC due to excessive compression efficiency loss. In particular, the hardware-based encoders studied in the past have a high priority for real-time operation and thus a great sacrifice in compression efficiency. Therefore, in this paper, the real time inter prediction hardware architecture that accelerates inter prediction and minimizes loss of coding efficiency of HEVC, is proposed. The bottom-up MV

prediction method proposed in this study greatly improves the accuracy of motion vector prediction by adopting the method of predicting the MV from the hierarchical adjacent PUs of the HEVC. As a result, the computational complexity of the IME is reduced by 67% without changing the coding efficiency. In this paper, the hardware-based IME capable of real-time operation by applying the bottom-up IME algorithm is proposed. Because the conventional hardware-based IME does not use the high-speed IME algorithm due to the occurrence of the idle cycle by the step dependency of the high-speed IME algorithm and the problem of accessing the reference data, the degradation of the coding efficiency is very large, more than several percent in Bjøntegaard delta bitrate (BD-BR). In this study, the TZS algorithm is adopted for the IME hardware which does not impair the computational complexity reduction performance of the TZS algorithm. In order to adopt the high speed IME algorithm in the IME hardware, the following three methods are proposed and applied to the IME hardware. First, the idle cycle, which is a persistent problem of high speed IME algorithm, is reduced by context switching of different reference pictures and different depth IMEs. Second, the multi bank SRAM structure using locality of reference data for quick and free access to reference data, is proposed. Third, to avoid the use of large amounts of switching muxes that result in overly free reference data access, a limited degree of freedom of reference data access based on the search center is proposed. As a result, the proposed IME hardware can process 4k UHD video at 60 fps using 4 reference pictures while supporting all block sizes of HEVC, and loss of coding efficiency is merely 0.11%. The use of hardware resources is 1.27M gates.

The newly adopted merge mode estimation in HEVC is a new coding tool that has excellent coding efficiency improvement. However, because the calculation complexity of merge mode estimation fluctuates with each PU, the hardware implementation for merge mode estimation is inefficient. Therefore, an efficient hardware-based MME method and hardware architecture are proposed. Because

the existing MME method determines whether the interpolation filter is applied by the neighboring PU, the utilization rate of the interpolation filter is 50% or less. Nevertheless, because the conventional hardware has been designed for the case using interpolation filters, there is a low-efficiency problem of hardware resources. In this paper, the MME hardware architecture with two datapaths that are reduced by half the vertical interpolation filter is proposed. In addition, the merge candidate allocation algorithm that minimizes loss of coding efficiency while maintaining high hardware utilization is proposed. As a result, the proposed hardware based MME achieves 7.4% faster execution time while using 24% less hardware resources than the conventional hardware-based MME. The proposed hardware based MME is capable of processing 4k UHD video at 30 fps with using 460.8K gates hardware resources.

Keywords : High-efficiency video coding, Video compression, Integer motion estimation, merge mode estimation, hardware organization

Student Number : 2013-30230